Sistemas Informáticos
Curso 2006-2007

e-QTI

Natalia Rivera Tolosa
Alberto Sánchez San Felipe
David Duce Pastor

Dirigido por:
Baltasar Fernández Manjón
Iván Martínez Ortiz

Facultad de Informática
Universidad Complutense de Madrid

1 Índice

2 Índice de figuras

3 Agradecimientos

Queremos agradecer la ayuda y comprensión de nuestras familias, que han vivido muy de cerca tanto los buenos momentos como los malos, porque gracias a su apoyo incondicional hemos podido llegar hasta aquí.

Agradecemos a Iván Martínez Ortiz, que no sólo ha dirigido, ayudado y colaborado en la elaboración de este proyecto, sino que nos ha entregado su conocimiento, su tiempo y su paciencia, y por la total disponibilidad que ha tenido durante el transcurso de este año.

También agradecemos al profesor Baltasar Fernández Manjón que ha tutelado este proyecto con eficacia, y sobre todo, agradecemos la confianza que ha depositado en nosotros.

Finalmente, agradecemos a todos nuestros compañeros y demás profesores que nos han acompañado en este largo camino, y muy especialmente a nuestros amigos, que nos han aconsejado y animado.

4 Autorización a la UCM

Los autores de este proyecto, Natalia Rivera Tolosa, Alberto Sánchez San Felipe y David Duce Pastor, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos (no comerciales) tanto la memoria como el código y el prototipo desarrollado.

Fdo. Natalia Rivera Tolosa

Fdo. Alberto Sánchez San Felipe

Fdo. David Duce Pastor

5 Resumen del proyecto

El objetivo de este proyecto es el desarrollo de un editor y motor de QTI (*Question and test interoperability*). QTI es un estándar que especifica la interoperabilidad de evaluaciones entre sistemas propuesta por IMS (http://www.imsglobal.org) y que describe la estructura básica para la representación de preguntas y sus correspondientes informes de resultados. La idea es poder reflejar todas las condiciones de un examen o auto evaluación en un fichero XML que pueda ser ejecutable por cualquier sistema compatible con QTI.

El proyecto tiene como objetivo principal crear un entorno de aula virtual que, mediante los lenguajes de marcado y las tecnologías Web asociadas, especifica e implementa un sistema de aprendizaje virtual de acuerdo al modelo de referencia IMS QTI. Esta aula virtual es flexible, abierta y ampliable de modo que se mejora la gestión, el acceso, la interactividad y la utilidad de la información educativa proporcionada mediante la red.

6 Abstract of the project

The aim of this project is to develop an engine and editor for QTI (*Question and test interoperability*). QTI is a standard that specifies the interoperability of evaluations among systems proposed by IMS (http://www.imsglobal.org), and it describes the basic structure for the representation of questions and its corresponding outcome reports. The idea is to be able to reproduce all the conditions of an exam or assessment in an XML file that might be executed in any compatible system with QTI.

The main purpose of the project is to create a virtual classroom environment that, using mark-up languages and Web technologies, specifies and implements a virtual learning system according to the IMS QTI model reference. This virtual classroom is flexible, open and extendable so that management, access, interactivity and the utility of the educational information provided by the network are improved

7 Palabras clave

- QTI
- JSF
- Ítem
- Test
- JSF
- Hibernate
- Spring
- J2EE
- MySQL
- XML

8 QTI

8.1 Introducción a QTI

La especificación IMS de Interoperabilidad de Preguntas y Tests (*Questions & Test Interoperability, QTI*) describe la estructura básica para la representación de preguntas (ítems) y tests (evaluaciones) y sus correspondientes informes de resultados. Por lo tanto, la especificación permite el intercambio de ítems, evaluaciones y resultados entre distintos sistemas de gestión de aprendizaje (*Learning Management Systems, LMS*), así como librerías de contenido y colecciones.

La especificación QTI está definida en XML para promover la máxima aceptación posible. XML es un lenguaje de marcado estándar muy flexible y potente, lo que permite que la especificación QTI sea extensible y personalizable, proporcionando una adaptación rápida incluso en sistemas especializados o propietarios.
La especificación QTI no limita el diseño de los productos especificando interfaces de usuario, paradigmas pedagógicos o estableciendo una tecnología que restrinja la innovación, la interoperabilidad o la reutilización.

8.2 Especificación de Casos de Uso

Los componentes de procesos (círculos), estructuras de datos (rectángulos) y los participantes (muñecos) de la arquitectura del sistema QTI se muestra en la siguiente figura:

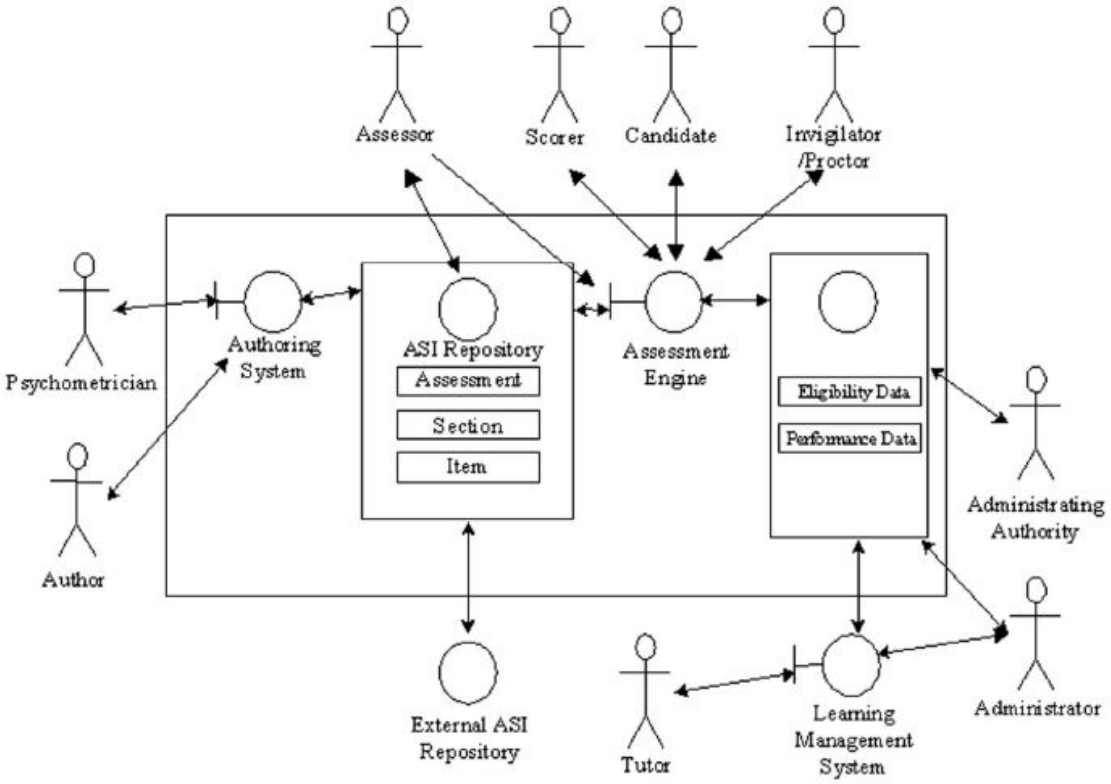


Ilustración 1: Representación de un sistema de Evaluación

Componentes clave de un sistema de evaluación:

- *Authoring system* (Sistema de Autoría) – proceso que soporta la creación y edición de las evaluaciones, secciones e ítems (ASIs).
- *Assessment engine* (Motor de Evaluación) – proceso que soporta la evaluación de las respuestas en términos de producción de ASIs relacionados con puntuaciones, evaluaciones y feedback.
- *Learning Management System* (Sistema de Gestión de Aprendizaje) – proceso/sistema que es responsable de la gestión de la arquitectura de aprendizaje.
- *Candidate data repository* (Repositorio de datos del candidato) – base de datos con la información específica del candidato.
- *ASI repository* (Repositorio de ASIs) – base de datos de los ASIs locales.
- *External ASI repository* (Repositorio externo de ASIs) – base de datos externa de ASIs que se importarán usando la especificación QTI.

Es posible un amplio rango de casos de uso pero sólo tres se presentan como ejemplo para esta especificación:

- Autoría: creación y edición de ASIs.
- Evaluación de alta participación: examen de candidatos.
- Evaluación de baja participación: apoyo del tutor usando ASIs.
- Evaluación basada en el contenido: contenido interactivo basado en QTI-XML.

8.2.1 Caso de uso de Autoría

La secuencia de procesamiento respecto a la estructura de datos ASI es la siguiente:

- El autor inicia el sistema de autoría.
- El autor crea o modifica ítems, secciones y/o evaluaciones. Éstos se exportan después usando la especificación QTI y se almacenan en una base de datos externa. Las estructuras de datos ASIs pueden consistir en un grupo complejo basado en múltiples evaluaciones y/o múltiples secciones recursivas y/o múltiples ítems.
- El autor puede importar ASIs que se usarán para crear nuevos ASIs. Estos ASIs importados también se ajustan a la especificación QTI.
- Una de las responsabilidades más importantes del autor es determinar el tipo de respuesta y asociar ésta a un tipo de presentación adecuado. Esta asociación dependerá del objetivo educativo del ítem. De la misma manera, el agrupamiento, selección y orden de secciones y/o ítems se basará en los objetivos educativos de la unidad ASI. El autor es responsable también de proveer la información específica de la vista del actor, que será importante porque ayudará a los usuarios a entender como se debe usar el material.

8.2.2 Caso de uso de Evaluación

8.2.2.1 Evaluación de alta participación

El proceso del motor de evaluación es el encargado de realizar esta actividad. Es importante notar que la operación interna del motor de evaluación está más allá del ámbito de esta especificación. Este caso se incluye porque justifica alguno de los componentes estructurales que se deben definir dentro de los ASIs. La secuencia de procesamiento del motor de evaluación es la siguiente:

- El evaluador (*assessor*) construye/selecciona los ASIs que se van a usar a lo largo del procedimiento de evaluación. Estos ASIs serán almacenados en una base de datos interna, por lo que la información de secuencia dinámica debe almacenarse en los propios ASIs.
- La evaluación se activa por el candidato y esta actividad la monitoriza el supervisor de exámenes (*proctor*). Las soluciones de los ASIs del candidato generan un conjunto de respuestas, de nuevo almacenadas internamente. Estas

respuestas son un conjunto de identificadores de los ítems junto con información asociada que caracteriza exactamente la respuesta.

- De forma síncrona o asíncrona, cada solución es evaluada por el proceso de respuestas para construir la puntuación inicial (la información de la puntuación es parte de la estructura de datos del ítem). Esta calificación requiere el uso de unas reglas de prueba (*evidence rules*) que se usan para definir los parámetros claves a través de los cuales las respuestas se van a evaluar. Esta evaluación del resultado del ítem se almacena en la estructura de datos de salida. Si un ítem se va a reutilizar en dos evaluaciones diferentes (por ejemplo, selección de alta participación o autoría de baja participación), entonces se puede usar el mismo contenido con diferente procesamiento de las respuestas y acumulación. En este caso, los sistemas de autoría serían los responsables de cambiar la descripción de la salida asociada y el procesamiento de las respuestas, así como los datos de acumulación y los parámetros.
- El proceso de acumulación analiza y recopila las salidas en términos de peso, etc., definido como parte de la estructura de datos de la sección. Esta información se almacena como parte del documento de evaluación.
- La fase final del proceso de evaluación es el proceso acumulado de evaluación en el que el documento de evaluación es procesado completamente usando las instrucciones de nivel de la estructura de datos de la evaluación.
- El paso final de este proceso es el feedback del documento de evaluación a la selección de actividad que puede tener como consecuencia una modificación de los ASIs presentados al candidato.

8.2.2.2 Evaluación de baja participación

El caso de uso del tutor es similar al de evaluación. Las diferencias son que el candidato recibirá feedbacks, incluyendo pistas y una o más posibles soluciones. La secuencia de procesamiento del motor de evaluación del tutor es la siguiente:

- El tutor construye/selecciona los ASIs que se van a usar a lo largo del procedimiento de tutoría. Estos ASIs serán almacenados en una base de datos interna, por lo que la información de secuencia dinámica debe almacenarse en los propios ASIs. Los candidatos podrían actuar con su propio tutor con parte de control sobre la actividad de selección.
- La sesión del tutor es activada por el candidato. El candidato responde a los ASIs y genera un conjunto de respuestas, de nuevo almacenadas internamente. Estas respuestas son un conjunto de identificadores de los ítems junto con información asociada que caracteriza exactamente cada respuesta.
- Cada respuesta es evaluada por el proceso de respuestas para construir la puntuación del ítem. Esta calificación requiere el uso de unas reglas de prueba que se usan para definir los parámetros claves a través de los cuales las respuestas se van a evaluar. Esta evaluación del resultado del ítem se almacena en la estructura de datos de salida. Esta información es usada después para generar feedbacks, por ejemplo, pistas o revelar un parte de la solución.

8.2.3 Caso de uso interactivo basado en el contenido

Es posible usar la especificación QTI para realizar cualquier tipo de material de aprendizaje. El contenido no tiene por que ser usado para un determinado tipo de evaluación. El proceso para desarrollar este contenido es el siguiente:

- Una herramienta de autoría se usa para construir el contenido y el diseño adecuados. Un asistente para la autoría se debería usar siempre que se requiera un formulario basado en preguntas, como por ejemplo una pregunta con respuestas múltiples. El autor debe crear el ítem por completo, incluyendo material de presentación, procesado de reacciones, feedbacks y descripciones de meta datos. Este material luego se exportará a su instancia equivalente QTI.
- Para el intercambio de contenidos interactivos entre LMS y el motor de presentación de contenidos se define el correspondiente modelo de interacción. Los diferentes componentes de QTI se asocian ahora en este modelo de transacción.

8.3 Preguntas, ítems y respuestas

8.3.1 Taxonomía de Respuestas

Las respuestas pueden ser clasificadas por tres tipos diferentes: Básicos, Compuestos y Patentado:

- Básico – es aquel que contiene un único tipo de respuesta.
- Compuesto – actúa como un contenedor de respuestas, es decir sería una combinación de tipos de respuestas básicas.
- Patentado – es un tipo alternativo soportado por la especificación QTI.

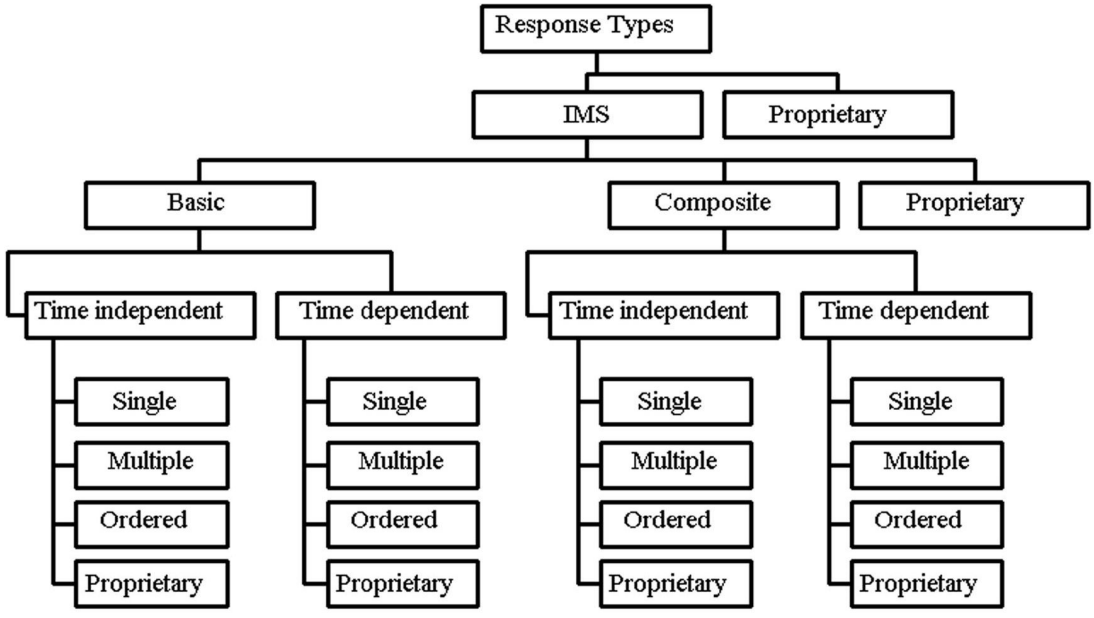


Ilustración 2: Taxonomía de Respuestas

Dentro de cada una de las clasificaciones, el tipo básico o compuesto se puede categorizar en dependiente del tiempo o no (*time dependent/ time independent*), dependiendo de si es importante o no el tiempo que se ha tomado el usuario desde que se le presento la pregunta hasta que se ha dado una respuesta**.**

Si es dependiente debe registrarse dicho tiempo para poder ser usado por los distintos tipos de respuestas y así establecer o definir una secuencia de eventos que el usuario ha de completar en un periodo de tiempo predefinido.

El último nivel de clasificación en la figura 21 esta basada en el número de acciones requeridas por el usuario.

- **Básico**:
  - Único – una única respuesta por parte del usuario.
  - Múltiple – una o mas respuestas del usuario.
  - Ordenado – una o más respuestas del usuario y el orden de las selecciones es significativo.
- **Compuesto**:
  - Único – una única respuesta por parte del usuario por cada ítem que forma el tipo de respuesta compuesto.
  - Múltiple – una o más respuestas por cada ítem que forma el tipo de respuesta compuesto.
  - Ordenado – una o más respuestas por cada ítem que forma el tipo de respuesta compuesto y el orden de las selecciones es significativo.

El siguiente nivel de taxonomía es mostrado en la figura 22

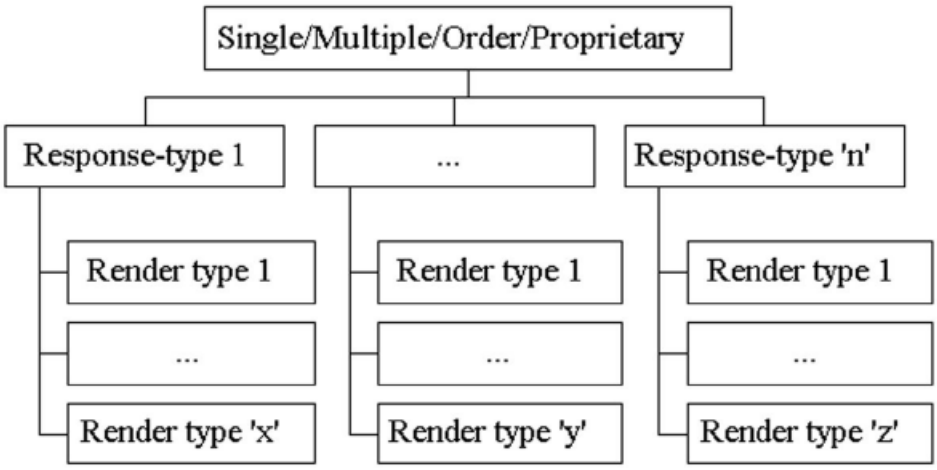


Ilustración 3: Relación entre tipos de respuestas y representación

Esta taxonomía muestra la relación existente entre los tipos de respuesta y los diferentes formatos de presentación. Por cada tipo de respuesta hay una o más formas de representar la pregunta, la respuesta y su selección. Por ejemplo puede ser usado simplemente una lista de opciones en formato texto, o una imagen con botones o huecos que pueden ser seleccionados.

Estas dos representaciones requieren la misma acción por parte del usuario, la identificación de la correcta información dentro de varias posibles opciones.

8.3.2 Tipos de ítems

8.3.2.1 Tipos de ítems básicos

Identificador lógico

Verdadero/falso estándar (opciones basadas en texto)

Interpretación basada en la elección.
Típica cuestión verdadero/falso de elección múltiple donde las posibles respuestas son formateadas de diferentes modos. El usuario debe seleccionar una de las opciones (verdadero-falso, si-no).

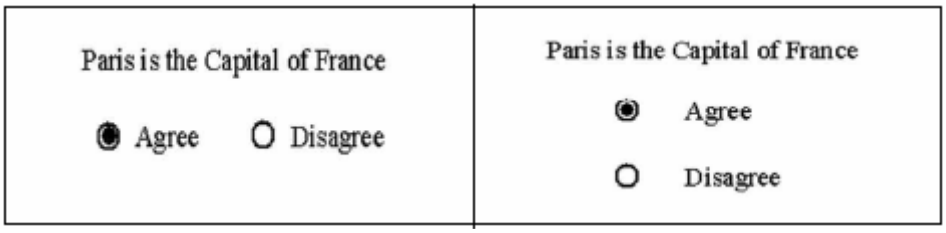


Ilustración 4: Verdadero/Falso estándar

Elección múltiple estándar (opciones basadas en texto)

Interpretación basada en la elección.
Típica cuestión de elección múltiple basada en texto. Se espera que el usuario elija una de las opciones disponibles pulsando el botón apropiado.

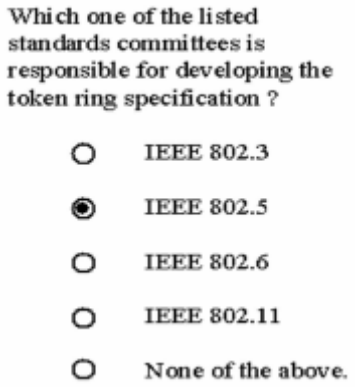


Ilustración 5: Elección Múltiple estándar basada en texto

Elección múltiple estándar (opciones basadas en imagen)

Interpretación basada en la elección.
Cuestión de elección múltiple basada en imagen. El usuario debe seleccionar una de las opciones pulsando en el botón apropiado.

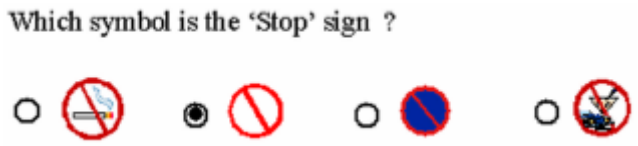


Ilustración 6: Elección Múltiple estándar basada en imágenes

Elección múltiple estándar (opciones basadas en audio)

Interpretación basada en la elección.
Cuestión de elección múltiple basada en audio. Se espera que el usuario haga su elección seleccionando el botón apropiado pero el audio solamente será activado pulsando en cada símbolo de la fuente de sonido. Los iconos usados para denotar los archivos de sonido dependen de la interpretación del sistema.

Ilustración 7: Elección Múltiple basada en audio

Respuesta múltiple estándar (opciones basadas en texto)

Interpretación basada en la elección.
Cuestión de respuesta múltiple típica. El usuario debe seleccionar cada una de las soluciones correctas usando los botones apropiados.

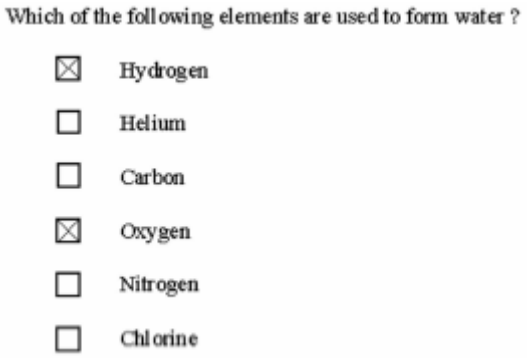


Ilustración 8: Respuesta Múltiple estándar basada en texto

Elección múltiple con una sola imagen (opciones basadas en imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de elección múltiple usando puntos seleccionables. Se espera que el usuario seleccione el botón apropiado.

Ilustración 9: Elección Múltiple basada en imagen

Respuesta múltiple con múltiples imágenes (opciones basadas en imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de respuesta múltiple usando varias imágenes con puntos calientes. El usuario debe usar el ratón para pulsar en este caso en las cuatro ruedas.

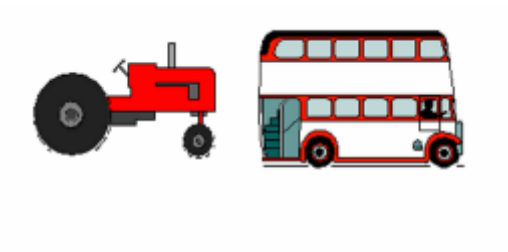


Ilustración 10: Respuestas Múltiples basadas en imagen

Elección múltiple (opciones basadas en barra de desplazamiento)

Interpretación basada en el deslizamiento.
Cuestión de elección múltiple usando puntos seleccionables con barra de desplazamiento. Se espera que el usuario mueva la barra de desplazamiento para señalar el valor correcto.

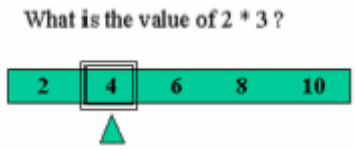


Ilustración 11: Elección Múltiple basada en barra de desplazamiento

Ordenar objetos estándar (objetos basados en texto)

Interpretación basada en los objetos.
Cuestión típica de objetos basados en el orden del texto. El usuario debe pulsar en cada objeto de texto y colocarlo en el orden correcto.

Ilustración 12: Ordenación de objetos estándar basados en texto

Ordenar objetos estándar (objetos basados en imagen)

Interpretación basada en los objetos.
Típico ordenamiento de objetos basados en imagen. Se espera que el usuario mueva cada objeto usando el ratón y moviéndolos alrededor de la pantalla.

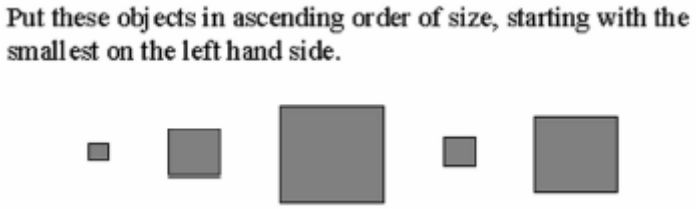


Ilustración 13: Ordenación de objetos basado en imágenes

Unir los puntos (basado en imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de conectar puntos. El usuario debe pulsar en las áreas apropiadas en la imagen y dibujar la correspondiente figura.

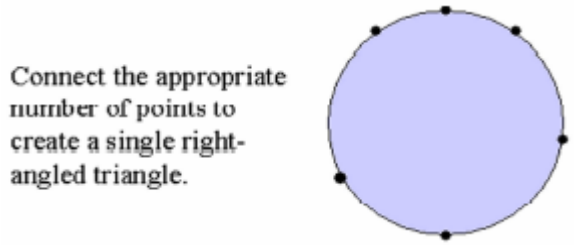


Ilustración 14: Unión de puntos basado en imágenes

Coordenadas XY

Imagen con zona seleccionable estándar (única imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de imagen con punto caliente. Se espera que el usuario pulse en el área apropiada de la imagen.

Ilustración 15: Imagen con zonas seleccionables

Unir los puntos (basado en imagen)

Interpretación basada en imagen con zona de puntos seleccionables.
Cuestión de conectar puntos. El usuario debe pulsar en el área apropiada en la imagen y dibujar la correspondiente figura.

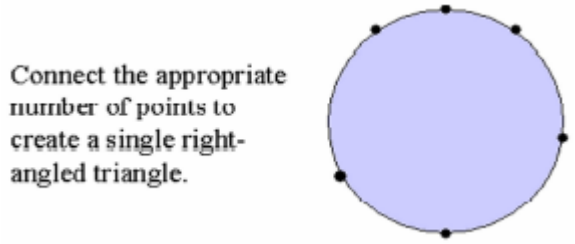


Ilustración 16: Unión de puntos los basado en imágenes

Cadenas

Rellenar un solo espacio en blanco estándar

Interpretación basada en rellenar huecos.
Típico texto de rellenar los huecos de un texto. Se espera que el usuario teclee la respuesta en el espacio destinado.

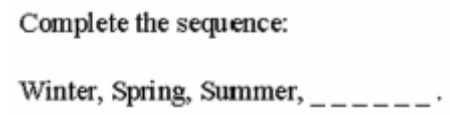


Ilustración 17: Rellenar un solo espacio en blanco estándar

Rellenar múltiples espacios en blanco estándar

Interpretación basada en rellenar huecos.
Texto para rellenar huecos con varias entradas. El usuario debe teclear las respuestas en los espacios propuestos para ello.

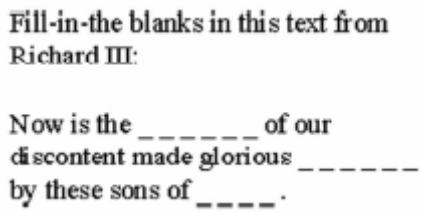


Ilustración 18: Rellenar múltiples espacios en blanco estándar

Respuestas cortas estándar (requerido texto)

Interpretación basada en rellenar huecos.
Típica cuestión de respuesta corta. Se espera que el usuario teclee el texto en el espacio suministrado.

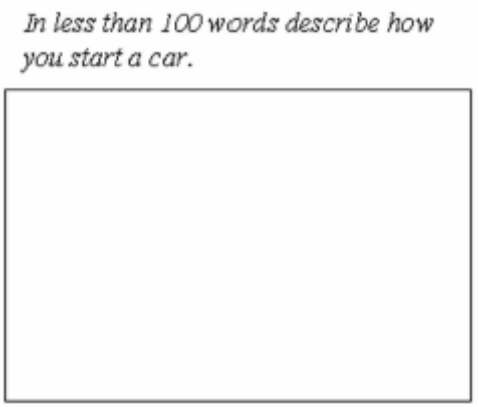


Ilustración 19: Respuesta corta estándar

Números

Rellenar huecos con números enteros o reales estándar

Interpretación basada en rellenar huecos.
Típica cuestión de rellenar huecos con números. El usuario debe escribir el número apropiado en la casilla suministrada.

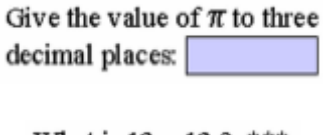


Ilustración 20: Rellenar huecos con números

Entrada numérica con barra de desplazamiento

Interpretación basada en barra de desplazamiento.
Cuestión para rellenar huecos numéricos con deslizador. Los usuarios deben mover la barra de desplazamiento hasta visualizar la respuesta necesaria.

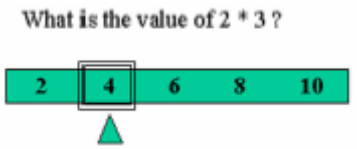


Ilustración 21: Entrada numérica con barra de desplazamiento

Grupos lógicos

Arrastrar y soltar estándar (imágenes múltiples)

Interpretación basada en objetos.
Cuestión típica de arrastrar y soltar. Se espera que el usuario pulse en la imagen de la respuesta apropiada y la desplace al lugar de respuesta apropiado.

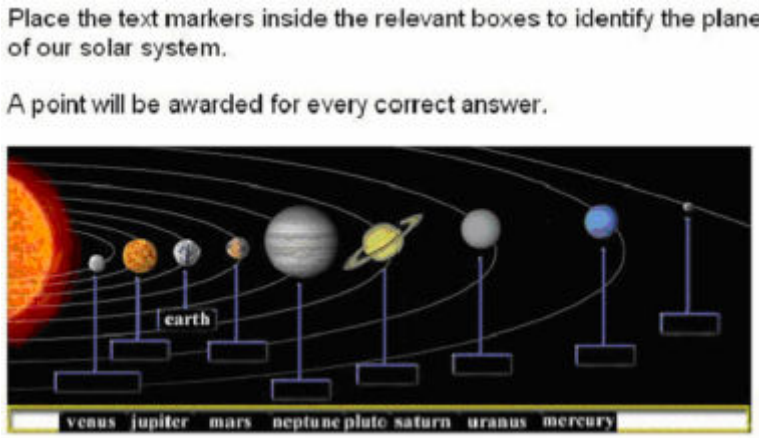


Ilustración 22: Arrastrar y soltar estándar basado en imágenes

Preguntas con asociación de respuestas

Este tipo de preguntas se caracterizan no sólo porque las respuestas sean las correctas sino por que la relación definida entre las distintas mismas sea la correcta

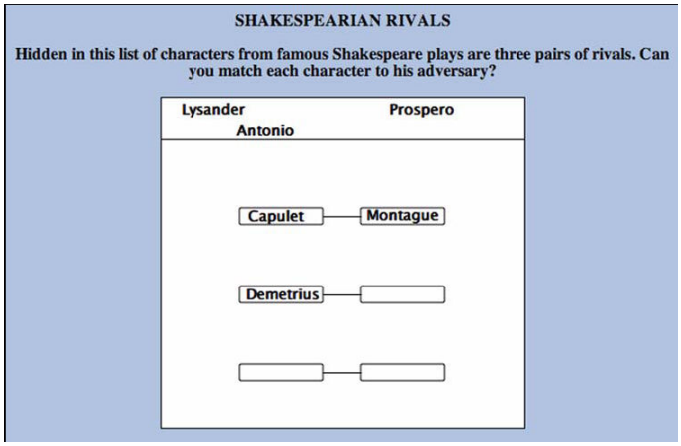


Ilustración 23: Shakespearian Rivals (Illustration).

8.3.2.2 Tipos de ítems compuestos

Elección múltiple con rellenar huecos

Elección múltiple con una pregunta adicional de rellenar huecos.
La respuesta basada en cadenas requiere la intervención humana para su corrección.

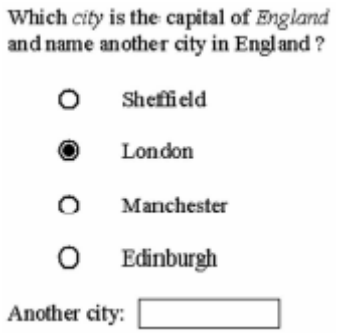


Ilustración 24: Elección múltiple con rellenar huecos

Respuesta múltiple basada en matriz

Cuestión compuesta de múltiples elecciones, ordenado en una matriz. Solamente se permite una respuesta por fila. El usuario debe elegir una opción por fila.

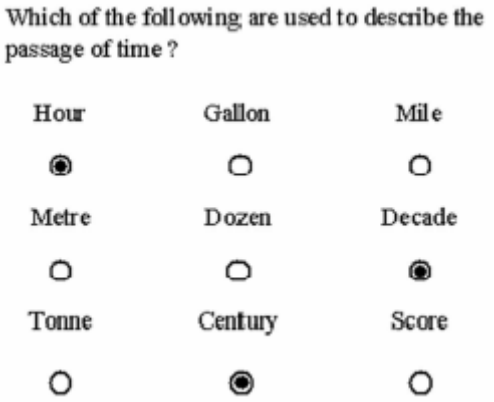


Ilustración 25: Respuesta múltiple basada en matriz

Marcación Múltiple basada en matriz

Consiste en que cada una de las respuestas propuestas puede pertenecer a varias preguntas

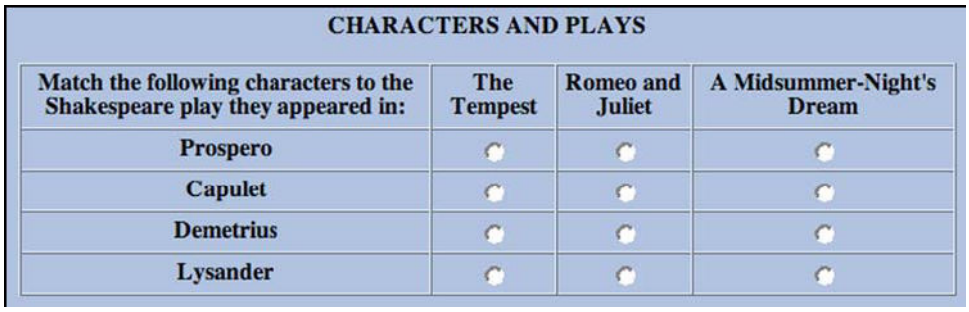


Ilustración 26: Characters and Plays (Illustration).

Respuesta múltiple basada en matriz para rellenar un texto

Consiste en la selección de una respuesta en una matriz para rellenar un texto con espacio en blancos

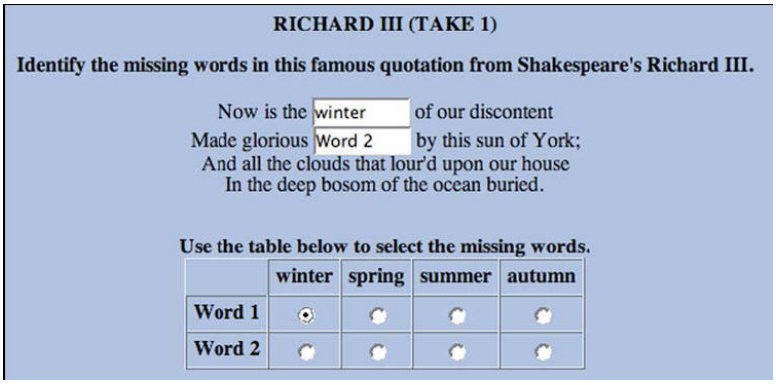


Ilustración 27: Richard III (Illustration 1).

Selección única mediante desplegables

El alumno deberá seleccionar la respuesta correcta de las que se le ofrecen mediante un desplegable.

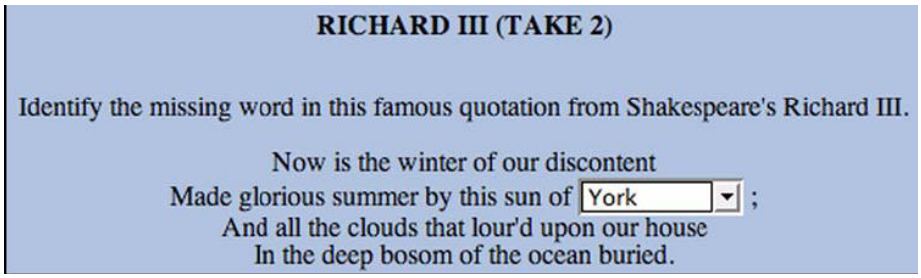


Ilustración 28: Ejemplo de selección múltiple mediante desplegable

Selección opcional

En este tipo de preguntas se le presentará al alumno varias respuestas para completar un texto y el alumno deberá decidir si es idóneo incluirlas y cual de ellas es la correcta.

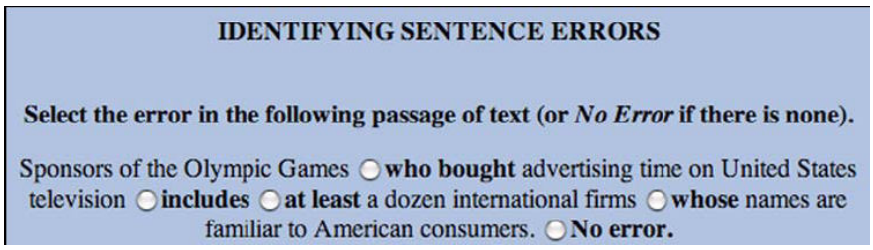


Ilustración 29: Ejemplo de selección opcional

Relación entre imágenes

Consiste en que el alumno relacione entre si determinados objetos de la imagen según una relación preestablecida

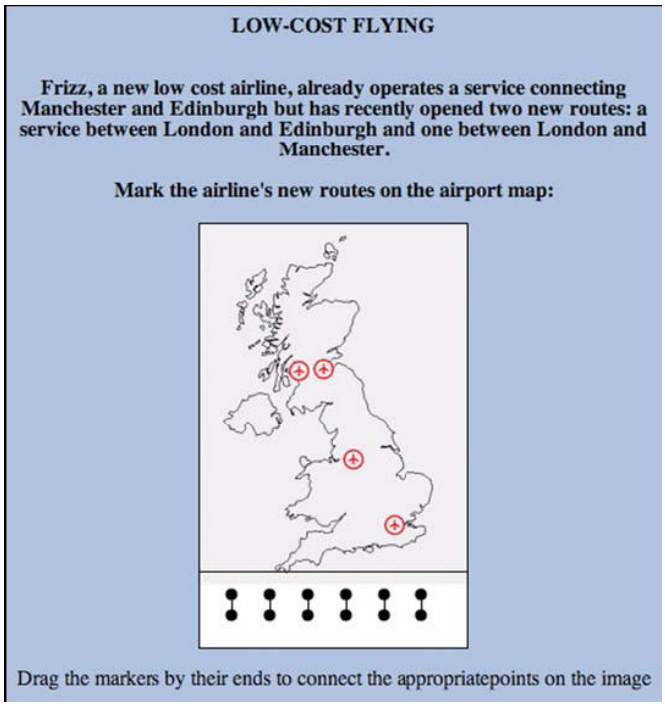


Ilustración 30: Relación entre imágenes 1

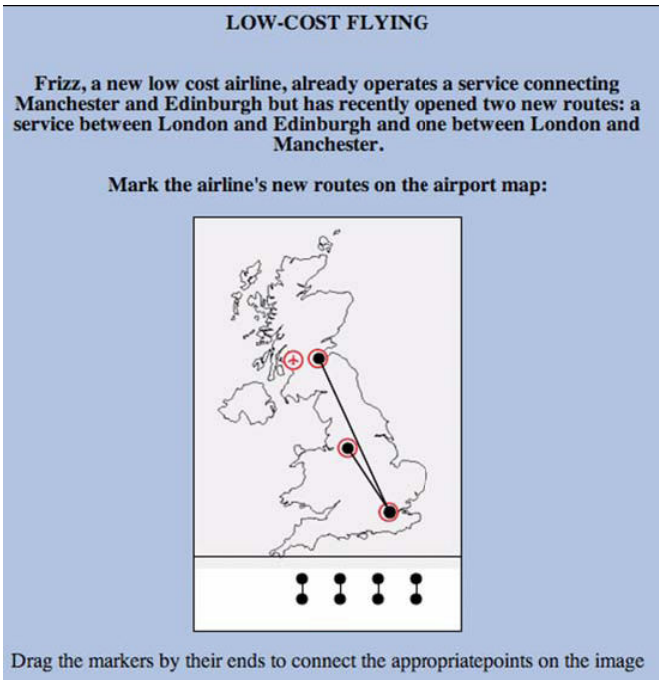


Ilustración 31: Relación entre imágenes 2

Relación etiquetas-imagen

El alumno deberá seleccionar la etiqueta correspondiente y asociarla mediante un clic a la imagen que le corresponda.

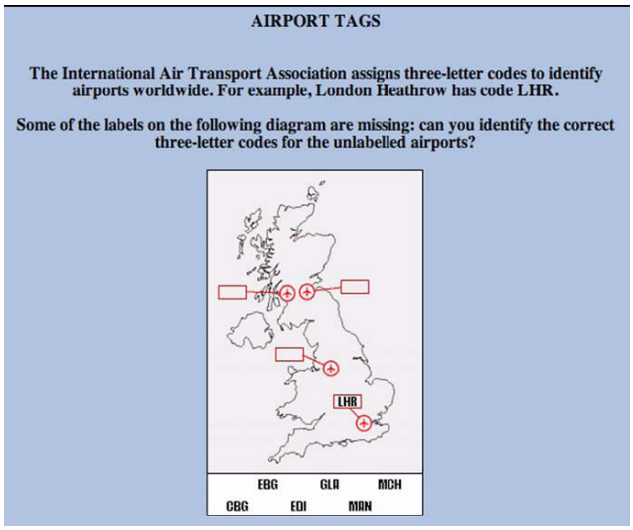


Ilustración 32: Relación etiquetas-imagenes

Selección de áreas acotadas

El alumno deberá colocar el item/ítems en el área acotada que le corresponda

Ilustración 33: Selección de áreas acotadas

8.3.2.3 Extensiones propietarias

Un requisito clave para esta especificación es su suporte para tipos de respuesta y representaciones propietarias. La manera de cómo se ajusta las extensiones propietarias con la taxonomía de tipos de respuestas esta mostrada en la figura 21.

Las extensiones propietarias pueden usarse como una alternativa al conjunto de los tipos IMS, a las clasificaciones básica o compuesta, o a las clasificaciones Única/Múltiple/Ordenada.

Las extensiones también se pueden encontrar en los tipos de representaciones y formatos mostrados en la figura 22.

8.3.2.4 Adaptative Ítems

Este tipo de preguntas consiste en la generación de las siguientes preguntas según la respuesta actual realizada por el usuario.

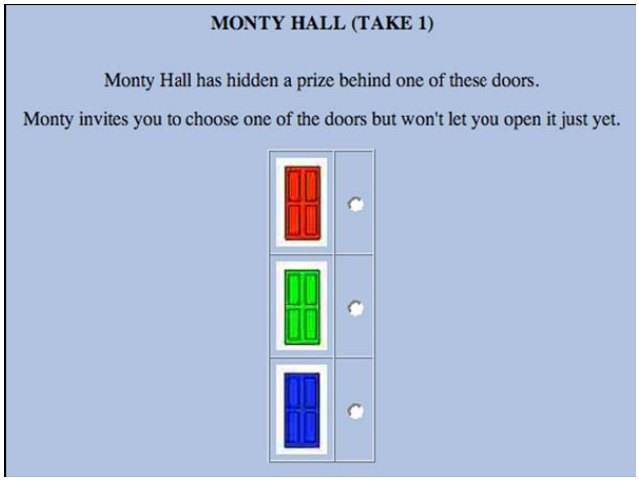


Ilustración 34: Monty Hall First Attempt (Illustration).

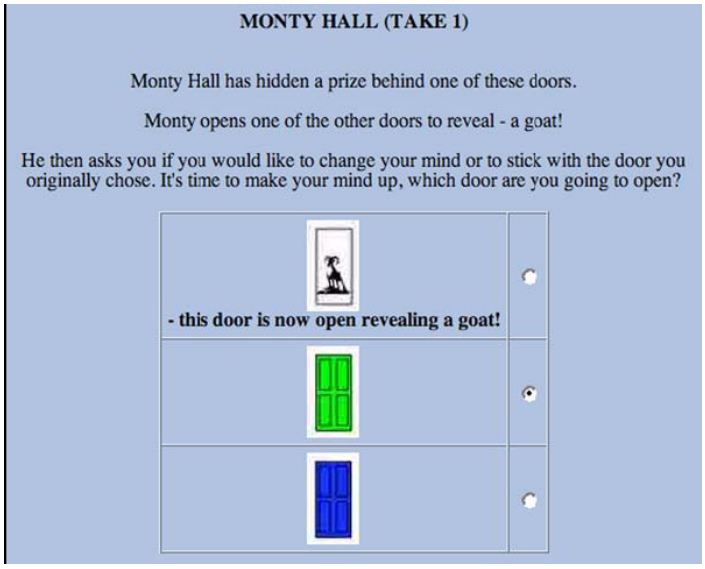


Ilustración 35: Monty Hall Second Attempt (Illustration).

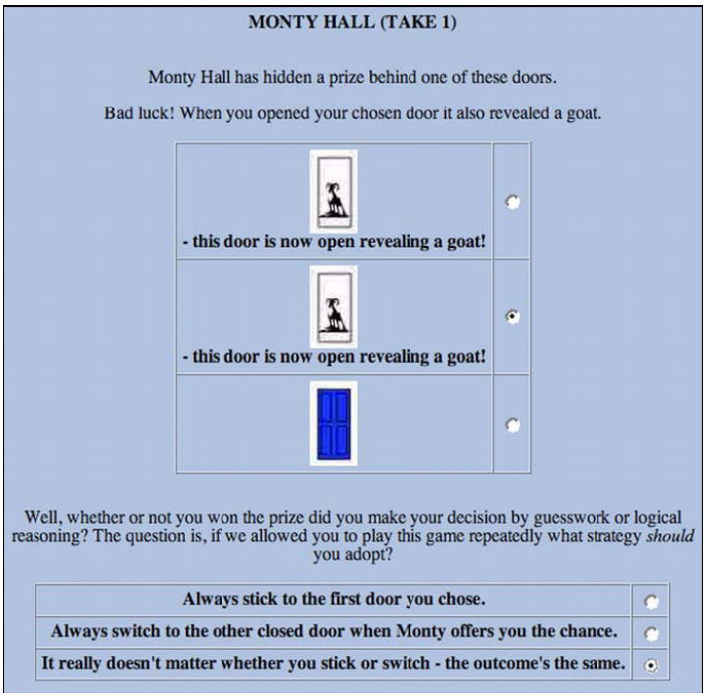


Ilustración 36: Monty Hall Third Attempt (Illustration).

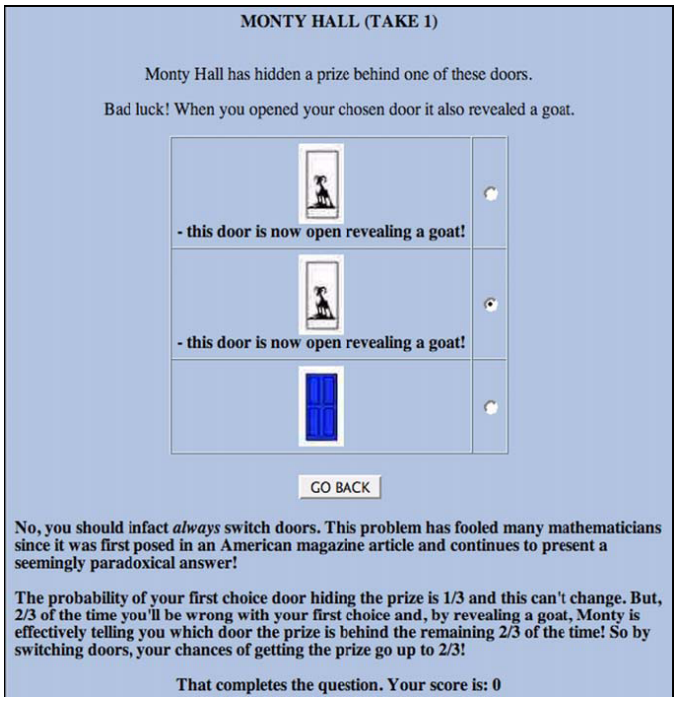


Ilustración 37: Monty Hall Final Feedback (Illustration).

8.3.2.5 Ítems Template

Este tipo permite la definición de una serie de plantillas para la generación de preguntas de similares características con pequeñas diferencias. El profesor crea las preguntas con los parámetros que desea que sean variables y define unas características de variación (Random(integer) por ejemplo) y es el sistema el que se encarga de la generación de las preguntas con los diferentes datos y respuestas para cada alumno.

8.4 Descripción general del Modelo de Datos.

8.4.1 Información básica.

Clase: *qtiMetadata*

Define una categoría de nivel superior que permite describir el perfil LOM y debe aparecer como un elemento directamente descendiente del objeto <lom>.
IMS no permite el enlace mediante XML de *qtiMetadata*, por lo que debe aparecer en paralelo con el objeto LOM como un objeto adicional de meta datos.

Esta clase contiene:

- *itemTemplate* (boolean [0..1]): Toma el valor *true* si el ítem actual se trata de un *itemTemplate*
- *timeDependent* (boolean [0..1]): Indica si el ítem es dependiente del tiempo o no.
- *composite* (boolean [0..1]): Toma el valor *true* si el ítem está compuesto por más de una interacción.
- *interactionType* [*]: Es el tipo de la interacción del ítem. Sus posibles valores son:
    - *associateInteraction*
    - *choiceInteraction*
    - *customInteraction*
    - *drawingInteraction*
    - *endAttemptInteraction*
    - *extendedTextInteraction*
    - *gapMatchInteraction*
    - *graphicAssociateInteraction*
    - *graphicGapMatchInteraction*
    - *graphicOrderInteraction*
    - *hotspotInteraction*
    - *hottextInteraction*
    - *inlineChoiceInteraction*
    - *matchInteraction*
    - *orderInteraction*
    - *positionObjectInteraction*
    - *selectPointInteraction*
    - *sliderInteraction*
    - *textEntryInteraction*
    - *uploadInteraction*

- *feedbackType* [0..1]: Describe, si existe, el tipo de *feedback* del ítem (pista, solución, o respuesta). Si está disponible, debe ser descrito como adaptativo o no-adaptativo, según lo sea el ítem o no. Un ítem adaptativo genera *feedback* a partir de todos los intentos anteriores, no sólo del último.
    - *none*

- *nonadaptive*
- *adaptive*

- *solutionAvailable* (boolean [0..1]): Toma el valor *true* si existe una solución disponible para el ítem.
- *toolName* (string256 [0..1]): Nombre de la herramienta utilizada para el objeto de evaluación.
- *toolVersion* (string256 [0..1]): Versión de la herramienta utilizada para el objeto de evaluación.
- *toolVendor* (string256 [0..1]): Nombre de la compañía que produce la herramienta utilizada para el objeto de evaluación.

El modelo de datos de QTI se muestra en la siguiente figura:

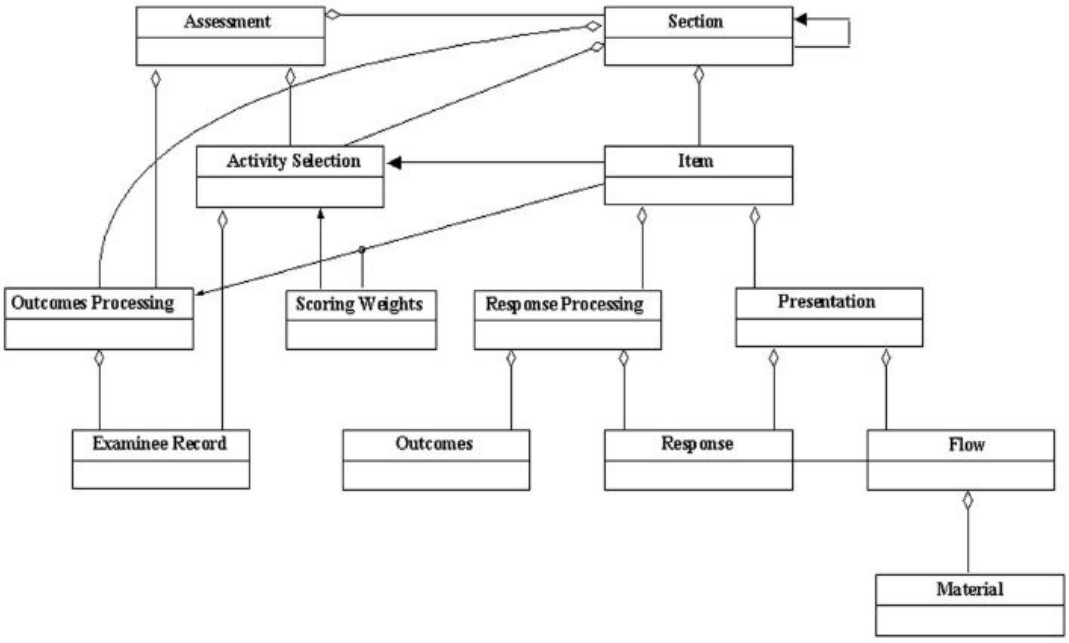


Ilustración 38: Modelo QTI

Los objetos en este modelo y sus claves de comportamiento son:

- *Assessment* (evaluación) – objeto que representa la estructura de datos *Assessment*.
- *Section* (sección) – objeto que representa la estructura de datos *Section*.
- *ítem* – objeto que representa la estructura de datos *ítem*.
- *Activity Selection* (selección de actividad) – selecciona la próxima actividad determinada por el progreso y los resultados obtenidos en el momento de seleccionar la actividad.
- *Outcomes Processing* (procesamiento de salidas) – procesa todas las evaluaciones de salida producidas para conseguir una evaluación general.
- *Scoring Weights* (Pesos de puntuación) – las puntuaciones ponderadas son asignadas a los resultados producidos en el procesamiento de la respuesta.
- *Response Processing* (procesamiento de respuestas) – procesamiento y evaluación de las respuestas de los usuarios.

- *Presentation* (presentación) – interpretación del contenido y posibles respuestas;
- *Examinee Record* (registro del examinado) – conjunto de resultados recopilados que son producidos durante el proceso completo. Esto es un registro permanente en el cual esta contenido el progreso histórico de los individuos.
- *Outcomes* (resultados) – conjunto de resultados que van a ser evaluados en la respuesta del objeto procesado. Esto determina las métricas de puntuación que van a ser aplicadas a las evaluaciones de respuesta.
- *Response* (respuesta) – respuestas que son suministradas por el usuario de los ítems, es decir, la entrada seleccionada por el usuario.
- *Flow* (flujo) – estructura de representación subyacente que define el bloque de relaciones entre los distintos componentes materiales;
- *Material* (material) – contenido que va a ser mostrado.

Esta estructura muestra la relación entre los tres objetos de datos centrales, Ítems, Secciones y Evaluaciones.

La siguiente figura muestra los tipos de objetos que pueden ser intercambiados:

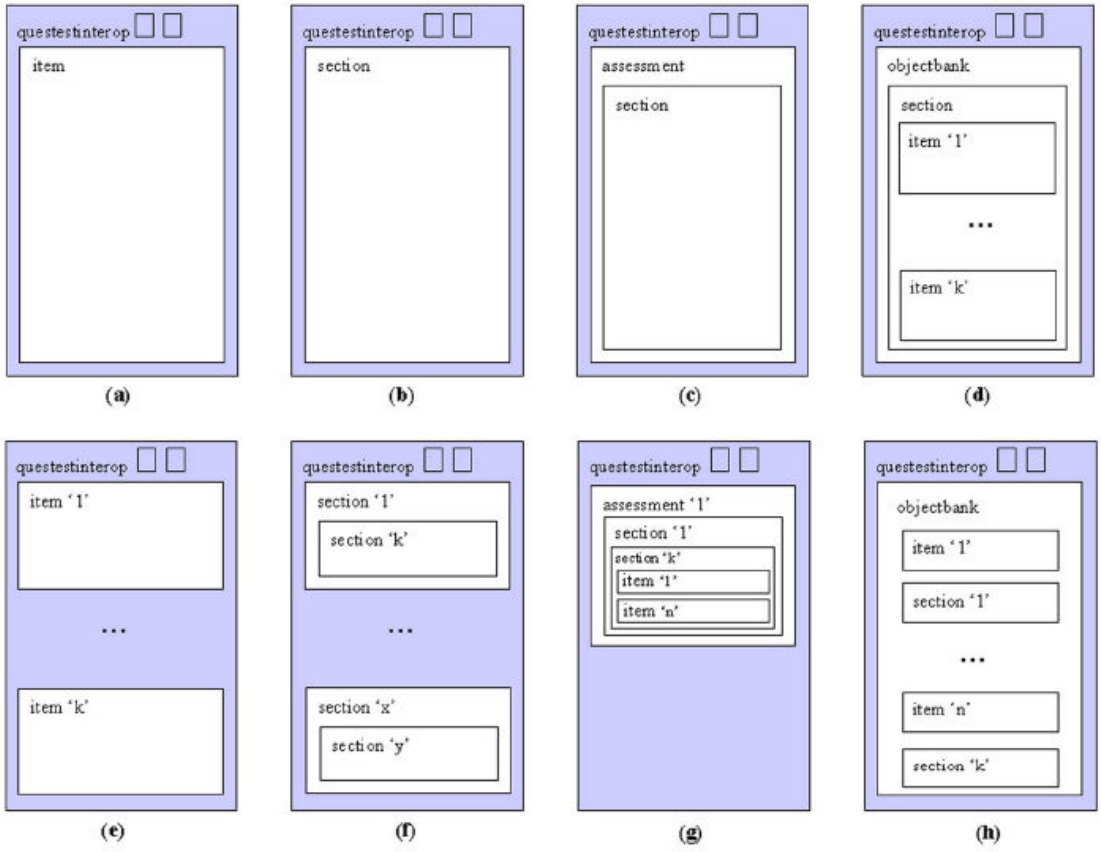


Ilustración 39: Objetos

Las relaciones entre los distintos objetos de datos se resumen de la siguiente forma:

- Una evaluación contiene al menos una sección (c)
- Una sección puede contener otras secciones (b) y (f)
- Un sección puede contener uno o más ítems (d) y (h)

- Un banco de objetos puede contener solamente ítems, solamente secciones o una mezcla de ítems y secciones.

La definición de la estructura de datos básica a nivel raíz es simple y completamente flexible. La estructura de datos puede usarse para importar/exportar estructuras de datos que posean:

- Una evaluación solamente (c) y (g)
- Una o más secciones solamente (b) y (f)
- Uno o más ítems solamente (a) y (e)
- Una evaluación puede o no contener más de una sección (c) y (g)
- Una sección puede o no contener ítems (b), (c), (f), y (g)
- Un banco de objetos (d) y (h)

Los bancos de objetos (*Object-banks*) son intercambiables mediante la definición de un tipo de paquete QTI, es decir, un conjunto de objetos de datos semejantes contenidos dentro del elemento <*questioninterop*>.

La siguiente figura muestra esquema en árbol genérico de XML:

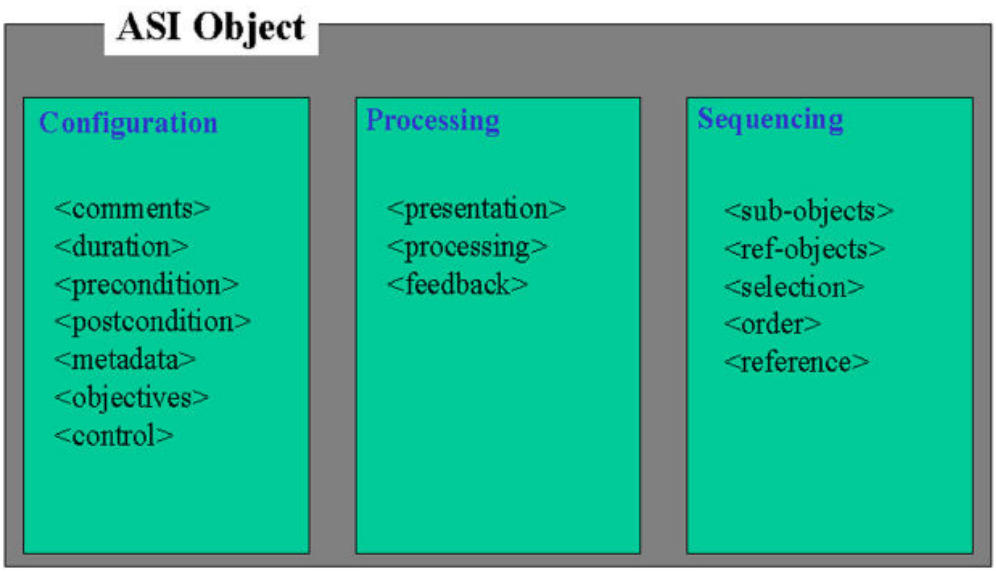


Ilustración 40: Árbol genérico de XML

Esta representación refleja la estructura de Ítem, Sección y Evaluación. Esta estructura tiene los tres componentes principales:

- *Configuration* (configuración) – generación del entorno apropiado para la correcta interpretación de la información contenida dentro del objeto.
- *Processing* (procesamiento) – procesamiento actual representado por el objeto, es decir, la representación de una cuestión y el correspondiente procesamiento de respuesta y correcciones.
- *Sequencing* (secuencia) – enlace, vínculo a los objetos referidos y la selección y secuenciamiento del próximo en ser procesado.

8.4.2 Evaluaciones. Secciones e ítems.

8.4.2.1 Ítems

questestinterop

Ilustración 41: questestinterop

El elemento *questestinterop* es el que contiene a todos los objetos QTI. Debe contener uno o más ítems.

Al tratarse del elemento central, debe aparecer únicamente una vez en todo el archivo XML.

No tiene atributos y su único elemento es el ítem.

ítem

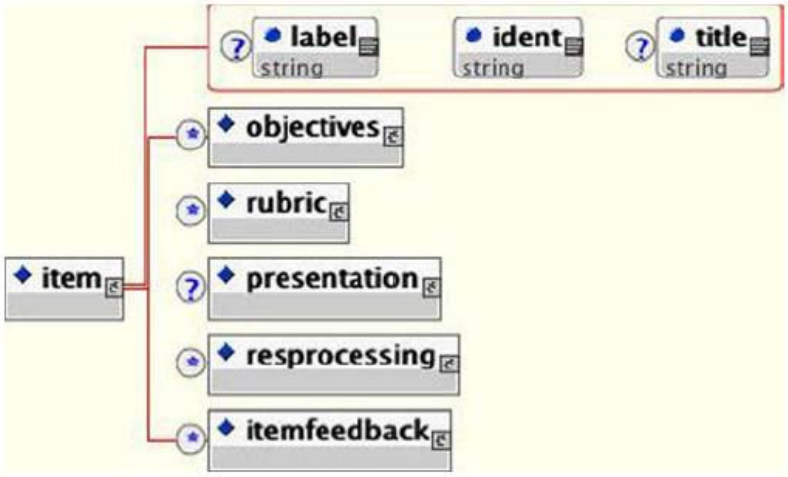


Ilustración 42: ítem

El elemento *ítem* es el único en la especificación QTI que puede ser intercambiado. Cada ítem está formado por cinco partes diferenciables:

- *objectives*: material utilizado para describir los objetos desde su vista correspondiente.
- *rubric*: material utilizado para definir el contexto del ítem disponible para cada vista diferente
- *presentation*: instrucciones para describir la naturaleza de la pregunta que va a ser presentada.
- *resprocesing*: instrucciones a tener en cuenta a la hora de evaluar las respuestas y crear la correspondiente puntuación y su *feedback*.
- *itemfeedback*: material de *feedback* presentado según la respuesta introducida.

Aparece una o más veces dentro del elemento *questestinterop*.

Contiene los siguientes elementos: *objectives, rubric, presentation, resprocessing* e *itemfeedback*. Y sus atributos son:

- *title*: (opcional) título del *ítem.*
- *label:* (opcional) etiqueta utilizada por el autor para identificar determinadas características mediante una palabra clave.
- *ident*: identificador único, a nivel global, del *ítem*.

Estructura general completa de un *ítem*:

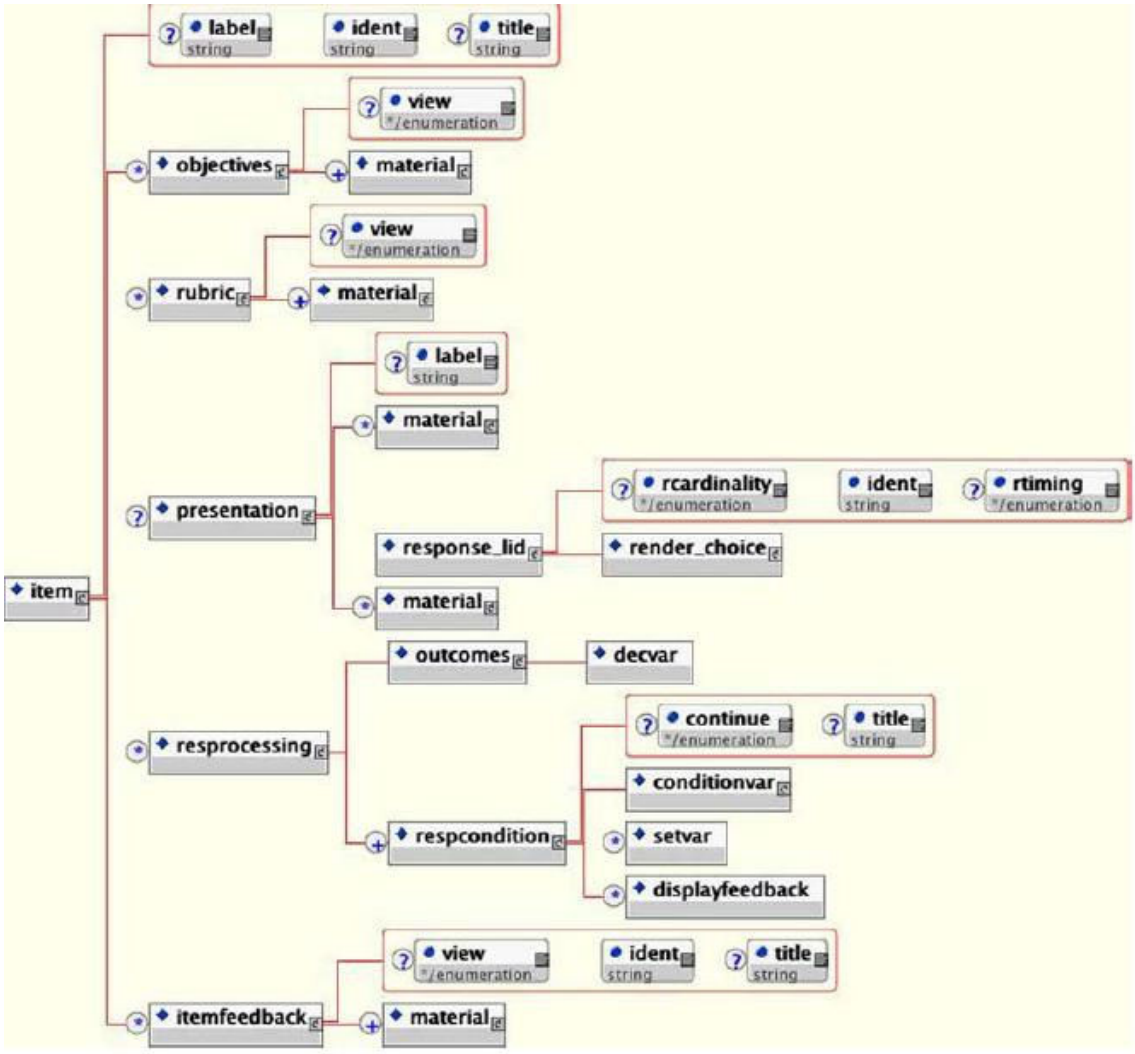


Ilustración 43: Estructura de un ítem

objectives

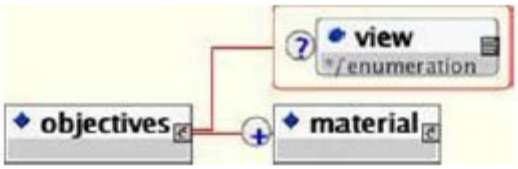


Ilustración 44: objectives

El elemento *objectives* debe ser usado para definir las objetivos del ítem para cada uno de los actores disponibles. Los objetivos no pueden incluir ningún tipo de contenido y así pueden ser presentados en un amplio rango de formas.

rubric

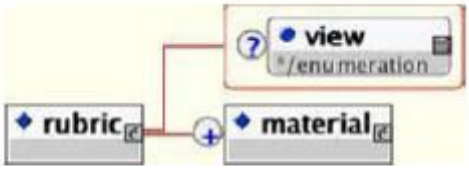


Ilustración 45: rubric

El elemento *rubric* debe usarse para presentar material contextual que es aplicado a un conjunto de ítems contenidos. Estas descripciones pueden ser suministradas para cada vista que es soportada. El elemento <*itemrubric*> es una alternativa soportada. El título para las vistas *All* y *Participant* debe ser siempre visualizado. En los casos donde hay demasiada información para ser visualizada se debería usar algún mecanismo de visualización por partes para permitir al participante acceder a la información ya lista, sin importar que página esté viendo.

Ambos elementos (*objectives* y *rubric*) aparecen cero o más veces dentro del elemento ítem.

Además, tienen un atributo *view*: lista numerada que indica el alcance de la información del elemento cuyos posibles valores que puede tomar son: *All, Administrador, AdminAuthority, Asesor, Autor, Candidate, InvigilatorProctor, Psycometrician, Scorer* y *Tutor,* siendo *All* su valor por defecto; y un elemento *material*.

presentation

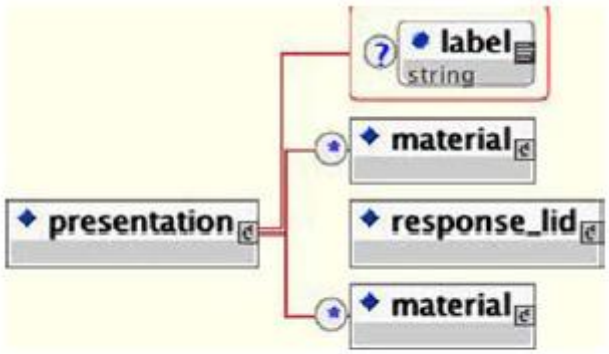


Ilustración 46: presentation

El elemento *presentation* contiene la información necesaria para la presentación de las preguntas durante el test. Incluye el *material* actual que debe ser presentado.
Tiene un atributo *label*, opcional y dos elementos: *material* y *respose-lid*.
También aparece cero o más veces dentro del elemento ítem.

response_lid

Ilustración 47: response_lid

El elemento *response_lid* contiene instrucciones para la representación de preguntas cuya solución está relacionada con la etiqueta de la respuesta marcada.
Aparece cero o más veces dentro del elemento *presentation*.

Sus atributos son:

- *ident*: identificador único para la representación de todo el bloque de la respuesta.
- *rcardinality*: (opcional) lista numerada que indica el número de respuestas esperadas por el usuario. Está definido como *rcardinality='Single'*.
- *rtiming*: (opcional) indica si la respuesta es dependiente del tiempo o no. Está definido como *rtming='No'*.

Tiene dos elementos: *material* y *render_choice*.

render_choice

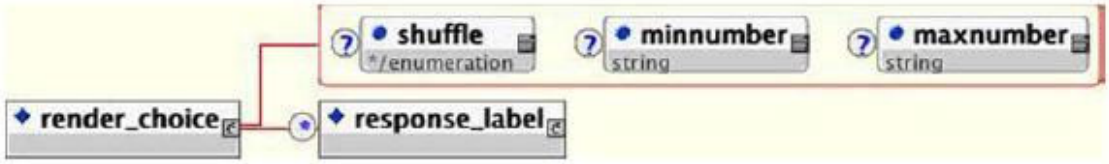


Ilustración 48: render_choice

El elemento *render_choice* muestra al motor cómo hacer que la pregunta tenga el formato de *multiple_choice*. El número de posibles respuestas viene determinado por el elemento <*response_label*>.

Aparece cero o más veces dentro del elemento *response_lid*.

Sus atributos son:

- *shuffle*: (opcional) lista numerada con valores *Yes* o *No*, que indica si la lista de respuestas puede ser visualizada por el usuario a medida que responde a las cuestiones. Su valor por defecto es *No*.
- *minnumber, maxnumber*: mínimo y máximo número, respectivamente, de respuestas que debe dar el usuario.

Tiene un elemento *response_label*.

response_label

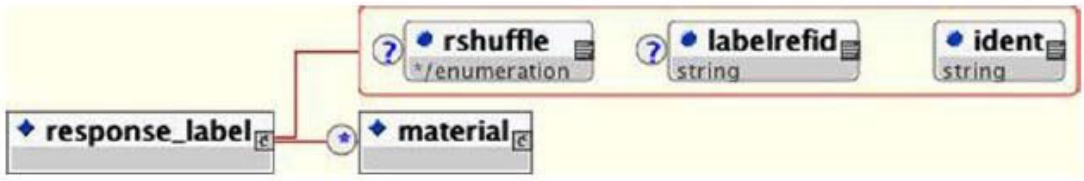


Ilustración 49: response_label

El elemento *response_label* define las posibles opciones de respuesta que se le presentan al usuario. Contiene información a cerca de la presentación que se le hace al usuario y es utilizado en el *response processing*.

Aparece cero o más veces dentro del elemento *render_choice*.

Sus atributos son:

- *labelrefid*: (opcional) etiqueta que puede ser empleada para la identificación de claves.
- *ident*: identificador único de la sección *response_label*. Es utilizado por el *respose processing* para la identificación de la respuesta seleccionada.
- *rshuffle*: (opcional) lista numerada con valores *Yes* o *No*, que indica si la lista de respuestas puede ser visualizada por el usuario a medida que responde a las cuestiones. Su valor por defecto es *No*.

Tiene un elemento *material*.

outcomes

Ilustración 50: outcomes

El elemento *outcomes* contiene la declaración de todas las variables disponibles para realizar el cálculo de las puntuaciones. Cada una de estas variables se declara mediante

un elemento *decvar* además de la variable por defecto *SCORE* cuyo tipo es entero y su valor por defecto es cero.

Aparece una sola vez dentro del elemento *resprocessing*.

No tiene atributos y contiene un elemento *decvar*.

decvar

El elemento *decvar* declara una única variable de puntuación, aparece una sola vez dentro del elemento *outcomes*. No contiene más elementos y sus atributos son:

- *varname*: (opcional) cadena de caracteres que indica el nombre que se le da a la variable declarada. Su valor por defecto es *SCORE*.
- v*artype*: lista de enteros que indica el tipo de la variable declarada.
- d*efaultvar*: (opcional) cadena de caracteres que indica el valor por defecto que debe dársele a la variable declarada.

respcondition

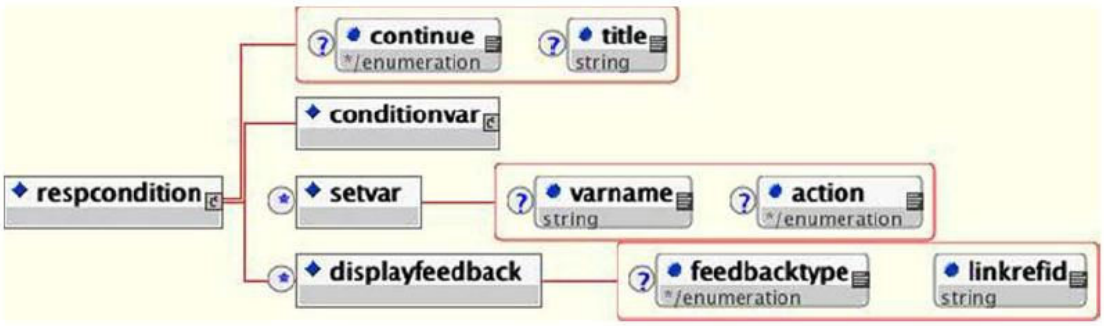


Ilustración 51: respcondition

El elemento *respcondition* contiene el test actual para verificar las respuestas correctas dadas por el usuario. Cada *respcondition* tiene un test actual, la asignación del valor que debe ser asociado a las variables de puntuación y la identificación del *feedback* que debe estar asociado a dicho test.

Aparece una o más veces dentro del elemento *resprocessing*.

Sus atributos son:

- *title*: (opcional) cadena de caracteres que indica la condición de la respuesta.
- *continue*: (opcional) lista de valores *Yes* o *No* que indica si deben ser aplicadas las condiciones de respuesta adicionales.

Tiene tres elementos: *conditionvar*, *setvar*, y *displayfeedback*.

setvar

El elemento *setvar* cambia el valor de las variables de puntuación para obtener el resultado asociado al *processing response* del test.

Aparece una vez dentro del elemento *respcondition*.

No tiene más elementos y sus atributos son:

- *varname*: (opcional) cadena de caracteres que indica el nombre que se le da a la variable declarada. Su valor por defecto es *SCORE*.
- *Action*: (opcional) lista numerada que indica la acción que se le aplicará a la variable nombrada. Su valor por defecto es *Set*.

displayfeedback

El elemento *displayfeedback* asigna el *feedback* correspondiente al *response processing* en caso de que el estado sea *True*.

Aparece cero o más veces dentro del elemento *respcondition*.

No tiene más elementos y sus atributos son:

- *Feedbacktype*: (opcional) lista numerada que indica el tipo de *feedback* que ha sido generado según las condiciones de las respuestas.
- *Linkrefid*: es el identificador del *feedback* asociado. Debe existir un elemento *itemfeedback* cuyo atributo *ident* coincida con este valor.

Conditionvar

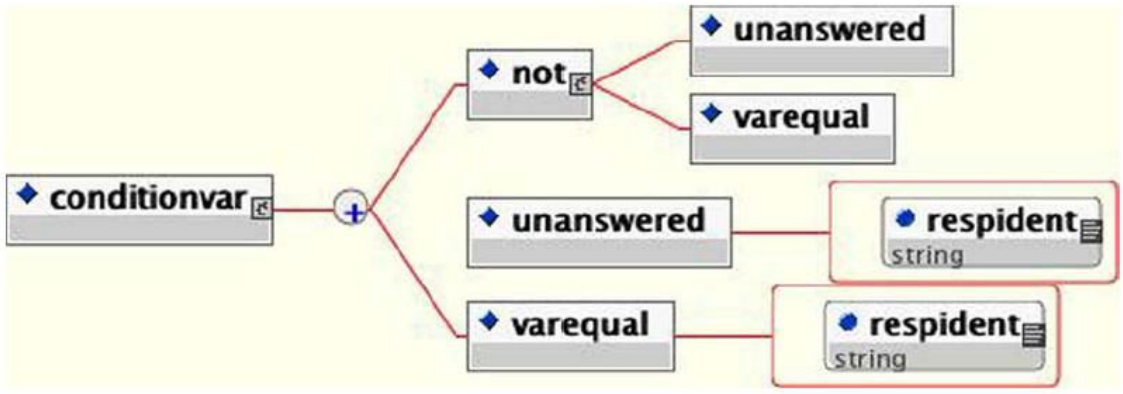


Ilustración 52: Conditionvar

El elemento *conditionvar* indica las condiciones de qué test serán aplicadas a las respuestas del usuario.

Aparece una vez dentro del elemento *respcondition*.

No tiene atributos y los elementos que contiene son:

Varequal

El elemento *varequal* es el test de equivalencia. Los datos del test se encuentran en la cadena PCDATA del elemento y debe coincidir con el atributo *ident* de uno de los *response_label*.

Aparece cero o una vez dentro del elemento *conditionvar* y del elemento *not*.

No contiene más elementos y dispone de un solo atributo:

- *Respident*: identificador del elemento *response_lid* designado madiante el atributo *ident*.

Not

El elemento *not* invierte el valor requerido de respuesta.

Aparece cero o más veces dentro del elemento *conditionvar*.

No tiene atributos, pero sí dos elementos: *unanswered* y *varequal.*

Unanswered

El elemento *unanswered* es la condición que se aplica en caso de que no se haya recibido respuesta de un ítem.

Aparece cero o más veces dentro del elemento *conditionvar* y del elemento *not*.

No contiene más elementos, y su único atributo es:

- *Respident*: corresponde al atributo *ident* del elemento *response_lid*.

Itemfeedback

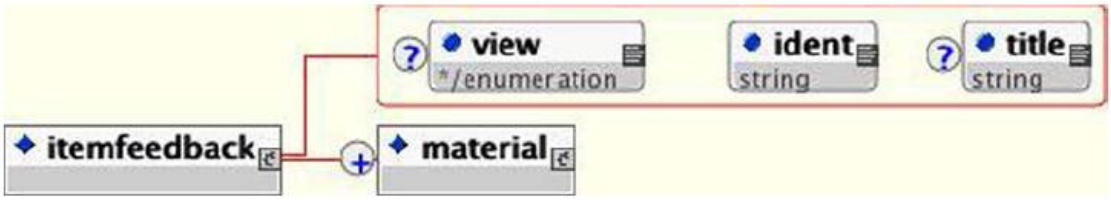


Ilustración 53: Itemfeedback

El elemento *itemfeedback* es un contenedor con la información que será presentada al usuario según sus respuestas.

Aparece cero o más veces dentro del elemento *item.*

Contiene un elemento *material* y sus atributos son:

- *Title*: (opcional) título de la sección de *feedback*.
- *Ident*: único identificador del *feedback*. Este identficador es usado en el elemento *resprocessing* para representar el *feedback* que se debe mostrar al usuario según las respuestas dadas.
- *view*: (opcional) lista numerada que indica el alcance de la información asociada, a cómo puede ser representado el elemento material. Los posibles valores que puede tomar son: *All, Administrador, AdminAuthority, Asesor, Autor, Candidate, InvigilatorProctor, Psycometrician, Scorer* y *Tutor,* siendo *All* su valor por defecto.

Material

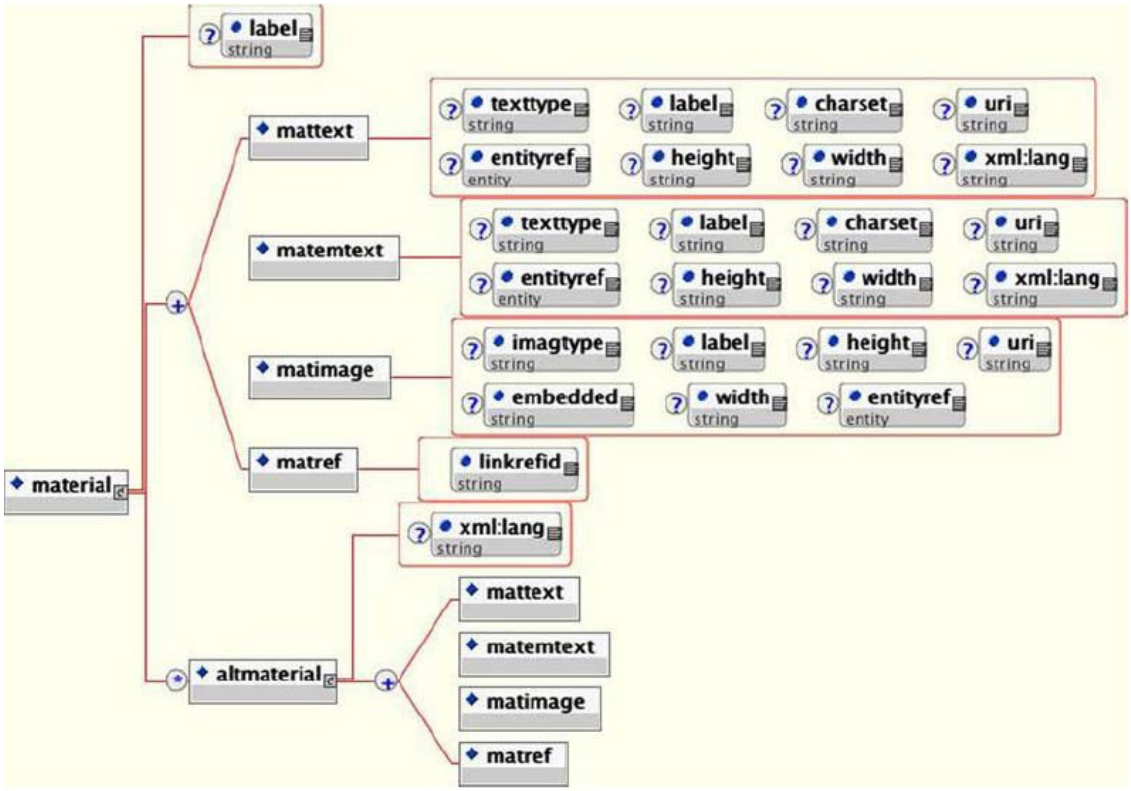


Ilustración 54: Material

El elemento *material* es un contenedor que almacena cualquier tipo de contenido que va a ser visualizado por el motor. Soporta únicamente tipos de texto e imágenes. Puede ser referenciado internamente para evitar la necesidad de hacer copias que dupliquen los datos. Puede contener también información adicional, que será presentada en caso de que el contenedor principal no pueda visualizarla.

Aparece una o más veces dentro de los elementos *objectives, rubric* e *itemfeedback.* Y cero o más en los elementos *presentation* y *response_label*.

Su único atributo es:

- *label:* (opcional) etiqueta utilizada por el autor para identificar determinadas características mediante una palabra clave.

Y contiene los siguientes elementos:

mattext

El elemento *mattext* contiene cualquier información que puede ser presentada al usuario.

Aparece cero o más veces dentro del elemento *material*.

No contiene elementos y sus atributos son:

- *texttype*: (opcional) formato del texto. Está definido según RFC1521. Su valor por defecto es "text/plain".
- *charset*: (opcional) grupo de caracteres que se usarán para la representación del texto. Su valor por defecto es "ascii-us".
- *label*: etiqueta utilizada como único identificador del texto.
- *uri*: identificador de recursos externos al texto que será presentado.
- *entityref*: (opcional) mecanismo alternativo para identificar las referencias externas que contienen el texto a presentar.
- *width*: (opcional) indica el ancho del texto.
- *heigth*: (opcional) indica el alto del texto.
- *xml:lang*: (opcional) idioma del texto. Sigue el estándar ISO639.

matemtext

El elemento *matemtext* contiene cualquier texto que debe presentarse resaltado al usuario. Su tipo depende del tipo del texto.

Aparece cero o más veces dentro del elemento *material*.

No contiene elementos y sus atributos son:

- *texttype*: (opcional) formato del texto. Está definido según RFC1521. Su valor por defecto es "text/plain".
- *charset*: (opcional) grupo de caracteres que se usarán para la representación del texto. Su valor por defecto es "ascii-us".
- *label*: etiqueta utilizada como único identificador del texto.
- *uri*: identificador de recursos externos al texto que será presentado.
- *entityref*: (opcional) mecanismo alternativo para identificar las referencias externas que contienen el texto a presentar.
- *width*: (opcional) indica el ancho del texto.
- *heigth*: (opcional) indica el alto del texto.
- *xml:lang*: (opcional) idioma del texto. Sigue el estándar ISO639.

matimage

El elemento *matimage* contiene información de las imágenes que serán presentadas al usuario.

Aparece cero o más veces dentro del elemento *material*.

No contiene elementos y sus atributos son:

- *imagtype*: (opcional) formato de la imagen. Está definido según RFC1521. Su valor por defecto es “image/jpeg”.
- *label*: etiqueta utilizada como único identificador del contenido de la imagen.
- *uri*: identificador de recursos externos a la imagen que será presentada.
- *entityref*: (opcional) mecanismo alternativo para identificar las referencias externas que contienen la imagen a presentar.
- *width*: (opcional) indica el ancho de la imagen.
- *heigth*: (opcional) indica el alto de la imagen.
- *embedded*: (opcional) define el tipo de codificación utilizada en caso de que la imagen haya sido colocada en una instancia XML.

matref

El elemento *matref* se utiliza para mantener referenciados los componentes *material*, por ejemplo *mattext*.

Aparece cero o más veces dentro del elemento *material*.

No contiene elementos y su único atributo es:

- *linkrefid*: es el identificador del contenido que está siendo referenciado. Será asignado por el atributo *label* del elemento asociado.

altmaterial

El elemento *altmaterial* es un contenedor de contenido alternativo. Su contenido será visualizado si, por alguna razón, el contenido primario no puede ser mostrado.

Aparece cero o más veces dentro del elemento *material*.

No tiene atributos, y contiene los siguientes elementos: *mattext, matemtext, matimage, matref*.

Grupos de elementos

Tipos de respuesta e interpretación

Hay cinco tipos básicos de respuesta. Es importante darse cuenta que el tipo de respuesta está determinado por la manera en la que es procesado internamente. Un único punto puede tener más de un tipo de respuesta, es decir, crear un tipo de respuesta compuesto. Si un nuevo tipo de respuesta es definido entonces puede ser añadido usando la extensión apropiada *response_extension*. El tipo clásico de pregunta de elección múltiple y respuesta múltiple puede ser soportado usando el elemento *response_lid*.

Hay cuatro tipos de interpretación: *render_choice, render_hotspot, render_slider* y *render_fib*. Es importante notar que el tipo de interpretación está impuesto sólo indirectamente por el tipo de respuesta pero está muy ligado al objetivo educacional de la pregunta. Si se identifica un nuevo tipo, entonces se puede añadir usando el elemento

de extensión apropiado *render_extension*. Los tipos clásicos de preguntas de elección múltiple o de respuesta múltiple pueden ser soportados usando *render_choice*.

El conjunto de tipos de respuesta e interpretación hace posible 20 combinaciones distintas. La inclusión de los atributos de cardinalidad y de tiempo hace que haya un total de 140 posibles combinaciones. No es posible examinar el tipo de respuesta, interpretación, cardinalidad y medición para asegurar automáticamente el tipo de la pregunta (si el tipo de la pregunta es una información requerida entonces debe ser suministrada usando el campo IMS QTI con la meta-data especifica *qmd_questiontype*). Esta ingeniería inversa no es posible porque algunas combinaciones pueden ser usadas para dar soporte a más de un tipo de preguntas.

Flows (Flujos)

Hay tres elementos que dan soporte a las capacidades de párrafos y bloques en QTI, llamado flujo, *flow_label* (etiqueta de flujo) y *flow_mat* (etiqueta de material).
La manera en la que el párrafo/bloque se presenta depende del motor de presentación pero se asume que estos motores de presentación serán capaces de procesar cada uno de estos elementos en los diferentes modos en que son combinados de un modo consistente. Se espera que este motor sea capaz de procesar flujos desde el contenido que no hace uso de ellos; no debería asumirse que todo motor sea capaz de tales diferenciaciones.

Itemfeedback (Corrección de Ítems)

El elemento *itemfeedback* contiene los elementos de pista y solución y sus contenidos se disparan usando el elemento *itemfeedback*. Este elemento puede contener varios conjuntos de consejos (pistas), soluciones y respuestas de corrección estándar. La diferenciación entre éstos se realiza usado el atributo *feedbacktype* y el *feedbackstyle*. El primero define el tipo de corrección que será mostrada (pista, solución, o respuesta) mientras que el último indica como el material de corrección será revelado, es decir, incrementalmente, etc.

Variable Manipulation (Manipulación de Variables)

La manipulación de las variables de puntuación declaradas en la combinación *outcomes*/*decvar* está contenida dentro del elemento *conditionvar*. La comparación de variables se hacen individualmente usando los elementos definidos como *varequeal*, etc. y el estado de estas comparaciones se puede invertir usando el elemento lógico NOT. El análisis del periodo de respuestas es soportado usando los elementos *durequal*, etc. (no se asume una variable por defecto asociada). La combinación de los elementos individuales *varequal*, etc. es posible usando dos técnicas:

- Implícita: la secuencia de elementos *varequal*, etc. dentro de un elemento *conditionvar* es por definición con una condición AND. El uso de múltiples elementos *conditionvar* es también tratado como una condición AND. La condición OR es conseguida a través del uso de múltiples elementos *respcondition*. La secuencia de estos es equivalente a una condición lógica inclusive OR.

- Explícita: el uso de los elementos lógicos AND y OR que combina los resultados de cada comparación por separado y los combina en una declaración de estado consolidado para el elemento *conditionvar*

Es recomendado que el enfoque implícito se use siempre que sea posible. Este enfoque proporciona un código más ínter operable.

Las variables de procesamiento de respuesta se declaran usando el elemento *decvar*. Cada implementación del IMS QTI debe generar una variable entera por defecto llamada SCORE cuyo valor por defecto es cero. Esta variable se usa siempre que un test condicional se aplica y el correspondiente *setvar* no incluye un nombre particular de variable. Cuando se soporta el procesamiento de una respuesta, hay dos condiciones especiales que necesitan ser tratadas:

- Cuando la respuesta no ha sido contestada – esto puede ser soportado usando dos técnicas, a saber, el elemento *unanswered* o el elemento *response_na*. El elemento *unanswered* es colocado dentro de *conditionvar* y es activado siempre que la respuesta no se haya intentado. El elemento *response_na* es una facilidad para la extensión propietaria que esta presente en cada una de la interpretación de elementos, es decir, *render_choise, render_hotspot,* etc.
- Cuando ninguna de las condiciones en el elemento *conditionvar* se satisface, se necesita declarar un estado por defecto. Este estado es capturado usando el elemento *other*. Cuando está incluido en *conditionvar* entonces se devolverá *verdadero* si ninguna otra condición se ha invocado.

8.4.2.2 Section

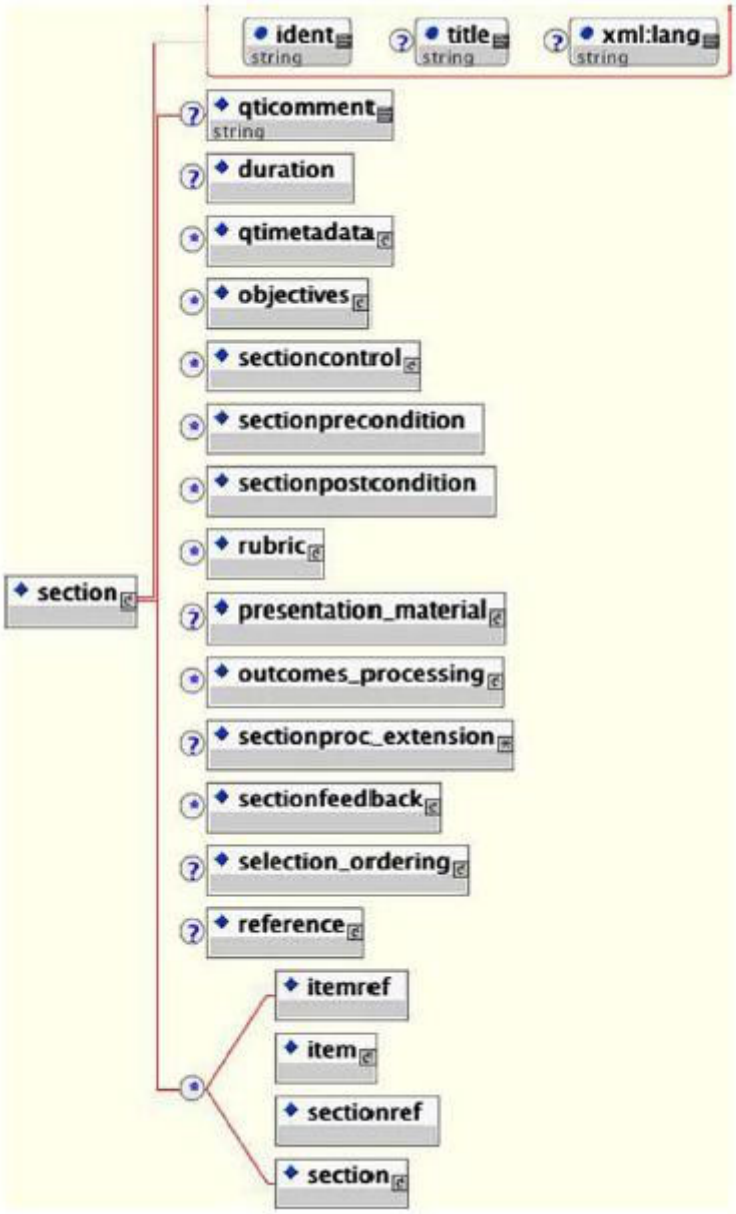


Ilustración 55: section

La estructura de datos *section* se utiliza para definir un árbol jerárquico complejo y estructuras de datos *ítem*.

Aparece una o más veces dentro del elemento *assessment*, cero o más en el elemento *questestinterop* y el element *section*.

Sus elementos y atributos son los siguientes:

metadata

Especificación sobre los propios datos para secciones se debe designar usando el vocabulario básico apropiado.

objectives

El elemento *objectives* debe ser usado para definir las objetivos de la sección para cada uno de los actores disponibles. Los objetivos no pueden incluir ningún tipo de contenido y así pueden ser presentados en un amplio rango de formas. El mecanismo por el cual los objetivos son visualizados va más allá del alcance de la especificación IMS QTI.

rubric

El elemento *rubric* debe ser usado para presentar material contextual que se aplica a un conjunto de secciones/ítems contenidas – esta información no debería ser usada para contener una cuestión actual o un componente de una cuestión. Estas descripciones pueden ser suministradas para cada vista que es soportada. El título para las vistas *All* y *Participant* debe ser siempre visualizado. En los casos donde hay demasiada información para ser presentada se debería usar algún mecanismo de visualización por partes para permitir al participante acceder a la información ya lista, sin importar que página esté viendo.

sectioncontrol

El elemento *sectioncontrol* debería ser usado para definir las condiciones por defecto para la exposición de tipos de correcciones de los usuarios. Las definiciones del nivel *Section* de *feedbackswitch*, *hintswitch* y *solutionswitch* tienen prioridad si no se encuentran definiciones en un nivel más bajo, es decir, dentro de un *ítem*. En caso de conflicto con una definición del nivel *Assessment,* las definiciones de nivel *Section* tienen prioridad.

presentation_material

El material contenido en este elemento debe usarse para almacenar información que sea relevante para cada uno de los objetos de evaluación contenidos en la sección. Este material debe ser visualizado para todos los participantes. No debería usarse para contener información como título y objetivos, puesto que hay elementos separados para este tipo de información.

outcomes_processing

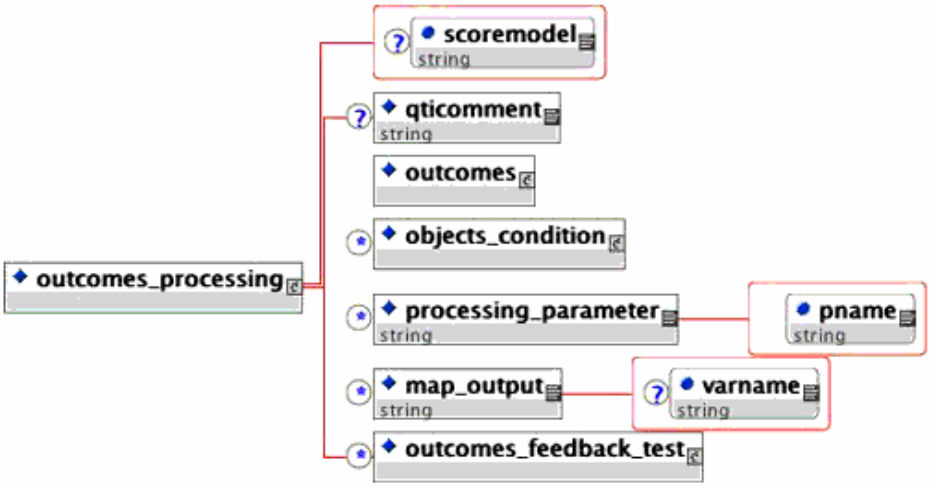


Ilustración 56: outcomes_processing

El elemento outcomes_processing es el lugar donde se procesan todos los resultados de las instrucciones para los Assessments y Sections. Los resultados obtenidos por el outcomes_processing son divulgados como los resultados de la Section o Assesment y sus variables deben ser únicas a través de todos los elementos outcomes_processing definidos por una section o assessment.

Su atributo es:

- scoremodel: Tipo String. Define el tipo del algoritmo que va a ser usado en el outcomes_processing para obtener los resultados de los assesments o sections que lo definen.

sectionfeedback

El elemento *sectionfeedback* puede almacenar contenidos que se usan para informar al participante de las calificaciones de sus respuestas, es decir, no puede ser usado para contener pistas y soluciones definidas dentro del contexto IMS QTI. La corrección se realiza a través del elemento *Outcomes_processing*.

reference

Material común en muchos objetos de evaluación dentro de objetos de evaluación, como por ejemplo un fragmento de texto, una imagen, etc. se almacena en este elemento. Este material es entonces referenciado desde dentro del elemento *material* apropiado. El material en este contenedor puede que no se visualice a menos que el elemento que referencia sea el mismo que se dispara.

Grupos de elementos

Selection & Ordering

La selección y secuenciamiento de secciones se soporta por medio de un mecanismo de preselección usando el elemento selection_ordering que define las secciones/ítems seleccionados desde el conjunto de Secciones disponibles y el orden en cada una de estas secciones/ítems se activan respectivamente.

Este proceso de selección y ordenamiento consta de tres pasos:

1. Objeto sequence: Contiene las reglas para el secuenciamiento y ordenamiento y evita que un objeto sea mostrado más de una vez.
2. Objeto selection: Este es el primer proceso por el cual se empiezan a aplicar las reglas de selección para los objetos, su uso esta limitado a los hijos directos del padre y se aplica en run-time.
3. Objeto ordering: Este es el segundo proceso por el cual se ordena los objetos ordenados según las reglas contenidas en el objeto sequence. Se produce este proceso en run-time.

8.4.2.3 Assessment

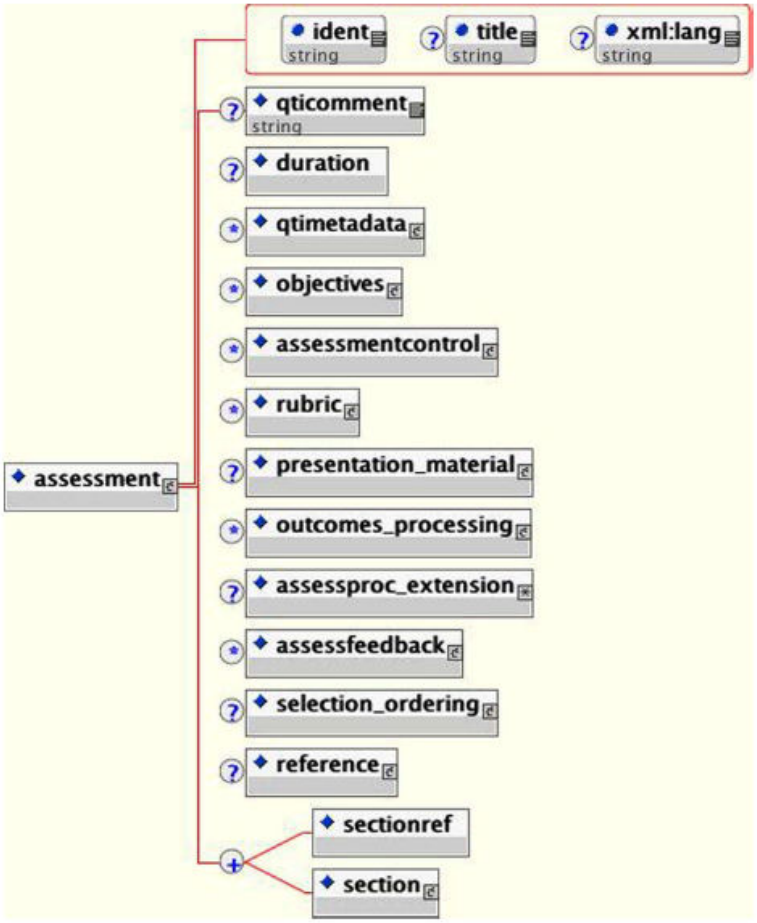


Ilustración 57: Assessment

La estructura de datos *assessment* contiene los intercambios de las estructuras de datos de test. Debe contener al menos un elemento *section*.

Aparece cero o más veces dentro del elemento *questestinterop*.

Sus elementos y atributos son los siguientes:

metadata

La especificación IMS QTI de *metadata* para la evaluación (*assessment*) se debe designar usando el vocabulario básico apropiado.

objectives

El elemento *objectives* debe usarse para definir los objetivos de la evaluación para cada uno de los actores disponibles. Los objetivos no pueden incluir ningún tipo de contenido, para que así se puedan presentar en un amplio rango de formas. El mecanismo por el cual los objetivos son visualizados va más allá del alcance de la especificación IMS QTI.

rubric

El elemento *rubric* debe ser usado para presentar material contextual que es aplicado a un conjunto de secciones/ítems contenidas – esta información no debería ser usada para contener una cuestión actual o un componente de una cuestión. Estas descripciones pueden ser suministradas para cada vista que es soportada. El título para las vistas *All* y *Participant* debe ser siempre visualizado. En los casos donde hay demasiada información para ser visualizada se debería usar algún mecanismo de visualización por partes para permitir al participante acceder a la información ya lista, sin importar que página esté viendo.

assessmentcontrol

El elemento *assessmentcontrol* debería ser usado para definir las condiciones por defecto para la presentación de tipos de corrección de los usuarios. Las definiciones del nivel *Assessment* de *backswitch*, *hintswitch* y *solutionswitch* tienen prioridad si no se encuentran definiciones en un nivel más bajo, es decir, dentro de una sección o un ítem. Esto significa que el nivel de definición *Assessment* actúa como estado por defecto.

presentation_material

El contenedor material en este elemento es usado para contener información que es relevante para cada uno de los objetos de evaluación contenidos en *Assessment*. Este material se debe visualizar para todos los participantes. No se debería usar para contener información como el título o los objetivos puesto que ya hay elementos separados para este tipo de información.

outcomes_processing

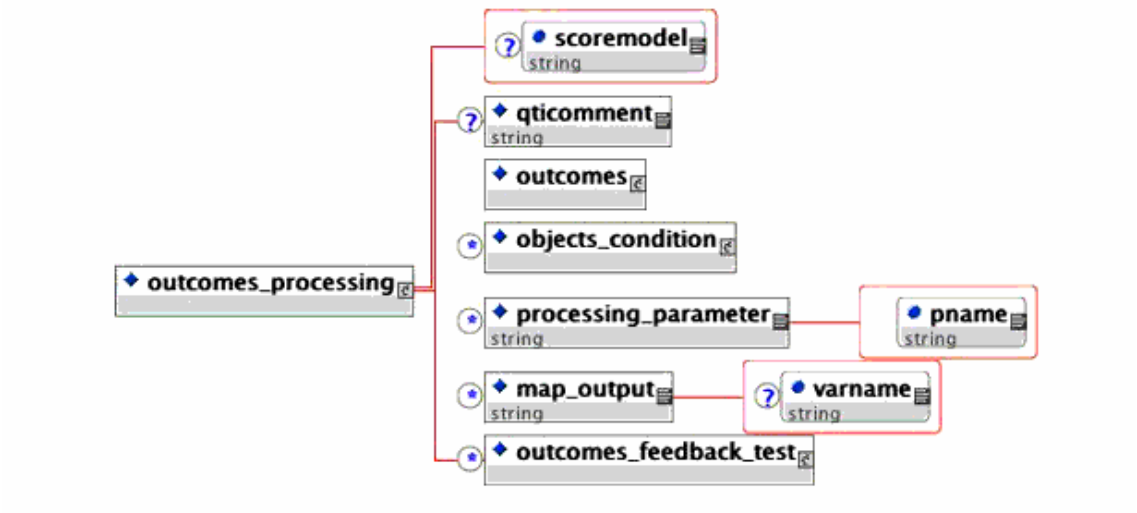


Ilustración 58: outcomes_processing

El elemento outcomes_processing es el lugar donde se procesan todos los resultados de las instrucciones para los Assessments y Sections. Los resultados obtenidos por el outcomes_processing son divulgados como los resultados de la Section o Assesment y

sus variables deben ser únicas a través de todos los elementos outcomes_processing definidos por una section o assessment.

Su atributo es:

- scoremodel: Tipo String. Define el tipo del algoritmo que va a ser usado en el outcomes_processing para obtener los resultados de los assesments o sections que lo definen.

assessfeedback

El elemento *assessfeedback* puede almacenar contenidos que se usan para informar al participante de las calificaciones de sus respuestas, es decir, no puede ser usado para contener pistas y soluciones definidas dentro del contexto IMS QTI. La corrección se realiza a través del elemento *Outcomes_processing*.

Reference

El material que es común en muchos objetos de evaluación dentro de objetos de evaluación, como por ejemplo un fragmento de texto, una imagen, etc. se almacena en este elemento. Este material es entonces referenciado desde dentro del elemento *material* apropiado. El material en este contenedor puede que no se visualice a menos que el elemento que referencia sea el mismo que se dispara.

Grupos de elementos

Selection & Ordering

La selección y secuenciamiento de evaluaciones se soporta por medio de un mecanismo de preselección usando el elemento *selection_ordering* que define las secciones/ítems seleccionados desde el conjunto de secciones disponibles y el orden en cada una de estas secciones/ítems se activa respectivamente.

Este proceso de selección y ordenamiento consta de tres pasos:

4. Objeto sequence: Contiene las reglas para el secuenciamiento y ordenamiento y evita que un objeto sea mostrado más de una vez.
5. Objeto selection: Este es el primer proceso por el cual se empiezan a aplicar las reglas de selección para los objetos, su uso esta limitado a los hijos directos del padre y se aplica en run-time.
6. Objeto ordering: Este es el segundo proceso por el cual se ordena los objetos ordenados según las reglas contenidas en el objeto sequence. Se produce este proceso en run-time.

8.4.3 Contenidos

8.4.3.1 Flujos

El siguiente ejemplo muestra la importancia de los flujos a través de la visualización de bloques múltiples de <*material*>.

Fill-in-the blanks in this text from
Richard III:

Now is the _ _ _ _ _ _ of our
discontent made glorious _ _ _ _ _ _
by these sons of _ _ _ _ .

Ilustración 59: Fill in blanks

En este ejemplo hay tres respuestas FIB (*fill in blank*) contenidas en un único ítem. Este texto y el texto de la pregunta introductoria son contenidos en varios elementos <*material*>. El problema es que no está clara la semántica de bloque definida para el elemento <*material*> y por tanto es confuso como la primera sentencia debe ser definida como un párrafo separado. Para resolver esta cuestión se introduce el concepto de flujo.

Un flujo es definido como un conjunto de contenido para ser gestionado por el motor de visualización como un bloque lógico o un párrafo. Un flujo puede contener otros flujos y por tanto se puede construir un sistema complejo de flujos jerárquicos. En el caso del ejemplo anterior tendremos dos flujos o bloques.

Cuando se utilizan estos flujos se ha de indicar mediante el elemento <*flow*> colocado dentro del elemento <*presentation*>. Con cualquier construcción de flujo se ha de usar <*flow*> dentro del elemento <*presentation*>. Los tres elementos que soportan flujos son:

- <*flow*> – indica el nivel superior de flujo dentro del elemento <*presentation*>
- <*flow_label*> – encapsula el elemento <*response_label*>
- <*flow_mat*> – encapsula el elemento <*material*>

Cualquiera de los tres casos los elementos puede ser recursivo, por ejemplo, <*flow_mat*> dentro de <*flow_mat*>, y las reglas de bloque deben ser definidas e implementadas consistentemente.

El uso de flujos permite controlar las cuestiones de diseño como por ejemplo:

- Párrafos para texto
- Mezclar texto y espacios para respuestas, como el tipo de respuesta FIB múltiple para un único ítem.
- Alineación de listas, incluyendo la alineación horizontal y vertical de las opciones disponibles en una pregunta múltiple elección.

Es recomendable que los ítems utilicen el enfoque de flujo. Permite una semántica más clara para controlar el diseño de los ítems y una mayor facilidad de soporte de técnicas

como XML *Style-sheets*. Sin embargo, esto no debe sobrecargar las cuestiones de estilo y que se oponga a nuestro principal objetivo de interoperabilidad funcional. Nuestra intención es permitir al autor influenciar en el diseño sin crear complicaciones en el motor de visualización, por ejemplo, diferentes fuentes de texto dará lugar a complicaciones significativas que han de ser evitadas permitiendo al motor de visualización usar su propio conjunto de valores por defecto.

8.4.3.2 Texto

La mayor cantidad de contenido a visualizar es de tipo texto. La especificación de QTI posee características para gestionar el contenido basado en texto, llamadas:

- *Mime type* – el valor por defecto es *text/plain*
- Conjunto de caracteres – indica al sistema la naturaleza del texto. El valor por defecto es *us-ascii*. En XML existe dos formatos de codificación: UTF-8 (por defecto) y UTF-16.
- Idioma – permite que el texto este disponible en una variedad de diferentes idiomas. Este mecanismo proporciona el contenido de un mismo ítem en uno o mas lenguajes utilizando <*almaterial*> por cada lenguaje dentro del ítem.
- Gestor de espacios en blanco – para ello ha de utilizarse el atributo *xml:space*. Por defecto no preserva los espacios en blanco.
- Énfasis – distingue partes de concretas de otras, se utiliza el elemento <*matemtext*>. La manera por la cual es obtenido, por ejemplo negrita, cursiva... se deja al motor de visualización.
- Párrafos – utiliza la estructura de flujo, el elemento <*matbreak*> que coloca una pausa en el material.
- Posición del texto – puede ser controlado por los atributos x_0, y_0 para indica la esquina superior izquierda del bloque. La altura y ancho del texto es controlado por los atributos *Height* y *Width*.

Es importante remarcar que las cuestiones estilísticas pertenecientes a la fuente del texto se encuentren fuera del alcance de la especificación. Estas cuestiones han de ser controladas usando hojas de estilo XSL, ficheros referenciados externamente como HTML, etc.

8.4.3.3 Imágenes

La presentación de imágenes requiere la especificación de un punto de referencia (*anchor point*). Este punto está definido por las coordenadas de la esquina superior izquierda en términos x_0, y_0. También es necesario otros dos atributos: *pixel-height* (altura) y *pixel-width* (anchura) de la imagen.

Un ejemplo de imágenes múltiples se encuentra en la siguiente figura. Cada imagen posee su propio tamaño y localización definido por x_0, y_0, anchura y altura. Los valores de los puntos x_0, y_0 se encuentra definidos respecto a la esquina superior de la pantalla. Los puntos seleccionables de cada imagen se especifican de la misma manera. En el

caso de coincidir las imágenes el orden de precedencia esta marcado por el orden de los elementos *response_label* (posee mayor precedencia el que se haya declarado antes).

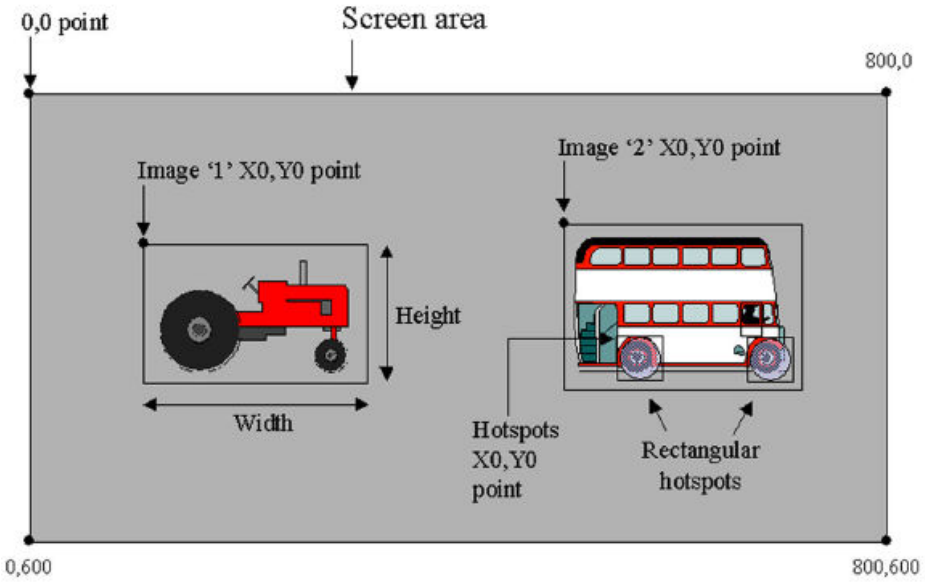


Ilustración 60: Screen area

8.5 La sesión de un ítem

La sesión de un ítem es la acumulación de todos los intentos realizados por un candidato en un ítem en particular. En algunos tipos de test, el mismo ítem puede ser presentado a un candidato múltiples veces. Cada ocurrencia o instancia del ítem es asociada con su propia sesión.

La clase que representa la sesión de un ítem (*itemSession*) es una clase abstracta que ayuda a ilustrar los requisitos de los procesos de reparto cuando se entreguen los ítems de los candidatos.

La sesión contiene el atributo *completionStatus* que almacena el estado de dicha sesión. Inicialmente contiene el valor "*not_attempted*" (el usuario no ha realizado ningún intento). Una vez que se ha producido el primer intento *completionStatus* toma el valor de "*unknown*" (valor desconocido). Permanece con este valor hasta que la sesión del ítem finalice o es modificado por *setOutcomeValue* durante el procesamiento de la respuesta. Hay cuatro posibles valores permitidos: *completed* (completado), *incomplete* (incompleto), *not_attempted* (ningún intento) y *unknown* (desconocido). La variable *completionStatus* puede cambiar a cualquiera de estos cuatro valores durante el procesamiento de la respuesta.

Si un ítem de tipo adaptativo asigna a la variable *completionStatus* el valor "*complete*" entonces la sesión ha de encontrarse en el estado *closed* (cerrado). Sin embargo, una sesión no tiene que esperar a la señal de *complete* para interrumpir la sesión, por ejemplo puede finalizar como respuesta a una petición directa del candidato, superar el tiempo máximo o por cualquier otra circunstancia excepcional. Los ítems de tipo No-adaptativo no requieren colocar un valor para *completionStatus*, mientras que los ítems de tipo Adaptativo deben mantener un valor adecuado y deberían establecer en *completionStatus* el valor "*complete*" para indicar cuando el ciclo de interacción, el procesamiento de las respuestas y los resultados de las evaluaciones deben parar.

La sesión almacena los valores que han sido asignados a todas las variables del ítem. Los valores de *completionStatus* y *duration* son tratados como variables de ítem especiales. Ellas comparten el mismo espacio de nombres como las variables de ítem que son explícitamente declaradas a través de las declaraciones de variable (*variableDeclarations*).

La sesión está asociada a un contexto (*sessionContext*) que proporciona información sobre el candidato, cuando y donde la sesión toma lugar, etc.

El siguiente diagrama ilustra los estados de la sesión de un ítem. No todos los estados pueden ser aplicados a cada escenario, por ejemplo el estado *Feedback* (corrección) puede que no exista para un ítem o no sea permitido en un contexto donde el ítem esta siendo utilizado. Asimismo, el candidato puede que no tenga permiso para revisar sus respuestas y/o examinar una solución. En la práctica, los sistemas pueden soportar solamente un número limitado de las transiciones de estados indicadas y/o soportar otras transiciones que no se encuentren aquí.

Un primer paso para los desarrolladores de sistemas es determinar qué requisitos de sus sistemas son identificados con los estados soportados en los mismos y emparejar las transiciones de estados indicadas en el diagrama con su propio modelo de eventos.

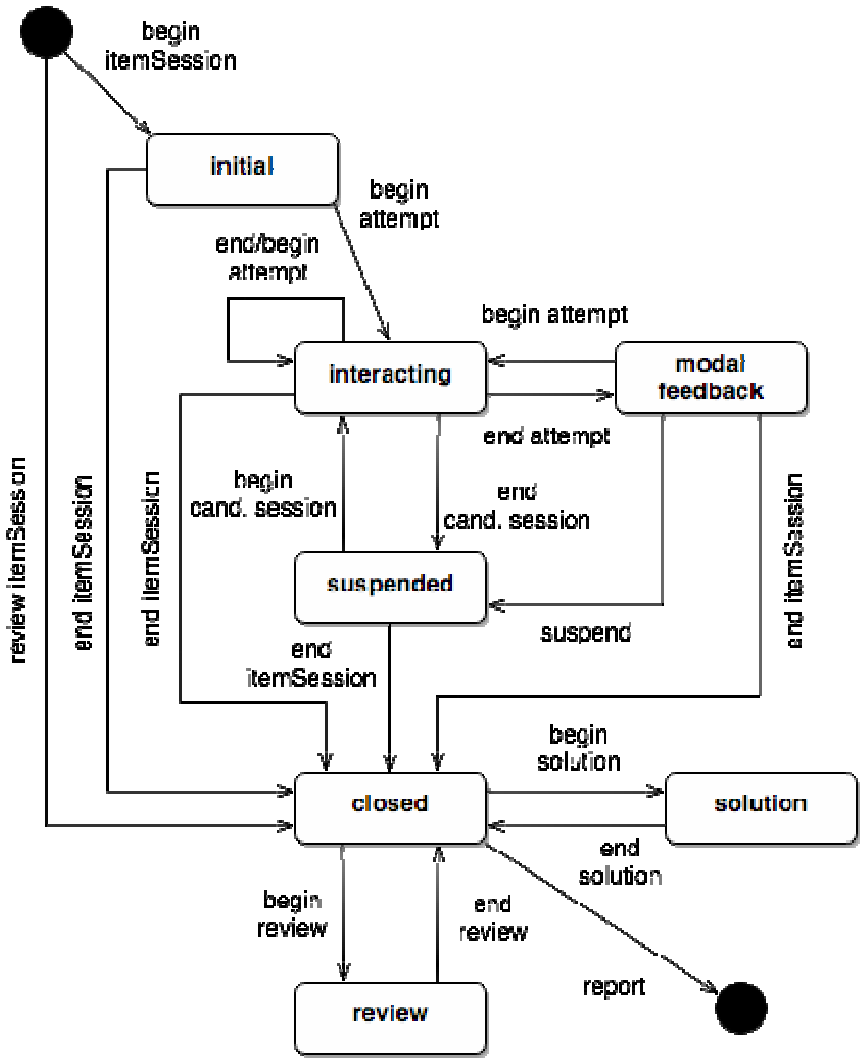


Ilustración 61: Ciclo de vida de la sesión de un ítem (itemSession)

La sesión comienza cuando el primer ítem asociado es elegible para poder ser entregado al candidato. El estado de la sesión del ítem es entonces mantenido y actualizado en respuesta a las acciones del candidato hasta que la sesión finalice. En cualquier momento el estado de la sesión se puede convertir en un itemResult. El sistema también puede permitir que un itemResult sea usado como la base para una nueva sesión permitiendo que las respuestas del candidato sean vistas en el contexto del propio ítem (y posiblemente compararlas con la solución) o incluso permitir que un candidato reanude una sesión interrumpida anteriormente.

El estado inicial de la sesión de un ítem representa el estado posterior a que se haya decidido que el ítem sea entregado al candidato pero anterior a dicha entrega.

En una evaluación no-adaptativa los ítems son seleccionados con antelación y el informe de las interacciones del candidato con todos los ítems se realiza al final de la evaluación, independientemente de que el candidato haya realizado algún intento para

todos los ítems o no. En realidad, se crean las sesiones para todos los ítems con el estado *'initial'* (inicial) y se mantienen en paralelo. En una evaluación de tipo adaptativa los ítems que estarán presentes son seleccionados durante la sesión basándose en las respuestas y salidas asociadas a los ítems presentados hasta el momento. Los ítems son seleccionados de un gran conjunto y el sistema solamente informa de las interacciones de los candidatos con los ítems que realmente sí han sido seleccionados.

La interacción de los candidatos con los ítems puede ser mediante ninguno o varios intentos. Durante cada intento el candidato interactúa con el ítem a través de una o más sesiones. Al final de la sesión del candidato el ítem puede pasar al estado *'suspended'* (suspendido) listo para la próxima sesión del candidato. Durante la sesión del candidato la sesión del ítem se encuentra en el estado *'interacting'* (interactuando). Una vez que el intento haya finalizado se realiza el procesamiento de la respuesta, después puede iniciarse un nuevo intento.

Para ítems no-adaptativos, el procesamiento de la respuesta normalmente sólo puede ser invocado un número limitado de veces, generalmente una única vez. Para ítems adaptativos, tal límite no es necesario porque el procesamiento de la respuesta adapta los valores que se asignan a las variables de salida basándose en los pasos a través del ítem. En ambos casos, cada invocación al procesamiento de respuestas indica el fin de cada intento. La apariencia del cuerpo del ítem, y cualquiera que sea la solución mostrada, está determinada por los valores de las variables de salida (*outcomeVariables*).

Cuando no se permitan más intentos la sesión del ítem pasa al estado de *'closed'* (cerrado). Una vez se encuentre en este estado los valores de las variables de salida son fijados. Un sistema de reparto aún debe permitir que el ítem sea presentado después de alcanzar el estado de '*closed'*. Este tipo de presentación se realiza en el estado '*review'* (revisión), las correcciones también pueden ser visibles en este punto si el procesamiento de respuestas ha tenido lugar y establece las variables de salida adecuadas.

Por último, la sesión del ítem podrá pasar al estado '*solution'* (solución) para los sistemas que soporten la visualización de las soluciones. En este estado, se reemplazará temporalmente las respuestas del candidato por los valores correctos obtenidos en las declaraciones de la respuesta (*responseDeclarations*) correspondientes o por el valor NULL si ninguna ha sido declarada.

Class: itemSessionControl

Cuenta con las clases asociadas testPart y sectionPart.

Cuando los ítems son referenciados como parte de un test, el test puede imponer restricciones en el número de intentos y en los estados permitidos. Estas restricciones se pueden especificar para ítems individuales, para una sección o para una testPart. Por defecto, los valores establecidos a nivel de testPart afecta a todos los ítems que contenga a no ser que esos valores se sobrescriban a nivel de assessmentSection o de forma individual en assessmentItemRef. Los valores por defecto sólo se usarán cuando no haya ninguna restricción aplicable.

Attibute: maxAttempts [0..1] : integer

Para ítems no-adaptativos, maxAttempts controla el número máximo de intentos permitidos en el contexto del test. Un valor 0 indicaría que no hay un límite establecido. Si no especificamos su valor será 1 para ítems no-adaptativos. Para ítems adaptativos, el valor de maxAttempts es ignorado, de modo que el número de intentos se limita con el valor de completionStatus de la variable de salida asociada.

Un valor de maxAttempts mayor que 1, por definición, indica que se puede aplicar alguna pista (feedback). Esto se aplica tanto en "Modal Feedback" como en "Integrated Feedback" siempre que sea posible. Sin embargo, una vez que se alcanza el número máximo permitido (o para ítems adaptativos, completionStatus se establece a completed) o no, la visualización de las pistas es controlada por las restricciones de showFeedback.

Attribute: showFeedback [0..1]: boolean

Esta restricción afecta a la visibilidad de las pistas después del último intento. Si su valor es false no se mostrarán. Esto afecta tanto en "Modal Feedback" como en "Integrated Feedback" incluso si el candidato tiene acceso al estado review. Por defecto su valor es false.

Attribute: showSolution[0..1]:bolean

Esta restricción indica si el usuario tiene acceso a la respuesta después de que se haya terminado la ejecución del ítem.

Attribute: allowReview [0..1]: boolean

Esta restricción se aplica sólo después del último intento. Si su valor es true la sesión del ítem puede pasar al estado review durante el cual el candidato puede revisar el itemBody con las respuestas que dio, pero sin poder modificarlas o reenviarlas. Si su valor es false el candidato no podrá revisar el itemBody o sus respuestas una vez que haya realizado su último intento. Por defecto su valor es true.

Si se permite el estado review, pero no se permiten las pistas, el sistema debe tener especial cuidado para no mostrar las pistas integradas que resultaron del último intento como parte del proceso de revisión.

Attribute: allowComment [0..1]: boolean

Algunos sistemas soportan que los candidatos añadan comentarios. Estos comentarios no se valorarán pero aportarán pistas del candidato hacia otros actores durante la evaluación. Esta restricción controla si el candidato puede o no añadir comentarios en el ítem durante la sesión.

Attribute: allowSkipping [0..1]: boolean = true

Un ítem se define como saltado si el candidato no ha proporcionado ninguna respuesta. En otras palabras, todas las variables de respuesta toman su valor por defecto o si no lo

tienen NULL. Esta definición es compatible con el operador numberResponded disponible en outcomeProcessing. Si es false, el candidato no puede saltarse el ítem, es decir, no puede continuar hasta que no proporcione un valor a alguna de las variables de respuesta. Por definición, un ítem sin variables de respuesta no se puede saltar. El valor de este atributo sólo es aplicable cuando el ítem está en un testPart en modo de presentación individual (individual submission). Note que si allowSkipping es true se debe contemplar que el candidato puede decidir no dar ninguna respuesta, por ejemplo, mediante la incorporación de un botón "skip".

Attribute: validateResponses [0..1]: boolean

Este atributo controla el comportamiento del motor cuando el candidato da una respuesta inválida. Una respuesta inválida se define como aquella que no satisface las condiciones impuestas por la interacción con la cual está asociada. Cuando validateResponse es true los candidatos no pueden enviar el ítem hasta que no den respuestas válidas para todas las interacciones. Cuando validateResponse es false el sistema puede aceptar respuestas inválidas. El valor de este atributo sólo es válido cuando el ítem está en un testPart en modo de presentación individual (individual submission).

8.6 El Procesamiento de la respuesta

El procesamiento de respuesta es aquel proceso por el cual el sistema de entregas asigna los resultados a la sesión del ítem basado en las respuestas del candidato. Los resultados pueden ser utilizados para proveer la corrección al candidato. La corrección puede ser entregada inmediatamente después de que finalice el intento del candidato o un tiempo más tarde, quizás como parte del informe sobre la sesión del ítem.

El fin de un intento, y por tanto del procesamiento de la respuesta, debe únicamente realizarse en una respuesta directa a una acción del usuario o en una respuesta a algún evento esperado, tales como el fin de una evaluación. Una sesión que entra al estado de suspendido quizás tenga valores para las variables de las respuestas que tienen todavía que ser enviadas para el procesamiento de la respuesta.

Para un ítem de tipo no-adaptativo se restaura los valores de las variables de salida a sus valores por defecto (o Null si no existe) antes de cada invocación al procesamiento de respuesta. Sin embargo, aunque el proceso de entrega puede invocar varias veces el procesamiento de la respuesta para un ítem no adaptativo únicamente debe informar sobre el primer conjunto de resultados producidos o limitar el número de intentos a algún límite predefinido acordado fuera del ámbito de esta especificación.

Para un ítem de tipo adaptativo no se restaura los valores de las variables de salida a sus valores por defecto. Un proceso de entrega que soporta los ítems adaptativos debe permitir al candidato revisar y entregar sus respuestas al procesamiento de corrección y debe solamente informar del último conjunto de resultados producidos. Además, debe presentar todas las correcciones aplicables e integradas al candidato. El procesamiento de respuesta posterior puede considerar que la corrección es conocida por el candidato cuando los resultados de la sesión son actualizados. Un ítem de tipo adaptativo puede indicar al proceso de entrega que el candidato ha completado la interacción y no se permiten mas intentos mediante asignando a la variable de salida *completionStatus* el valor complete (completo).

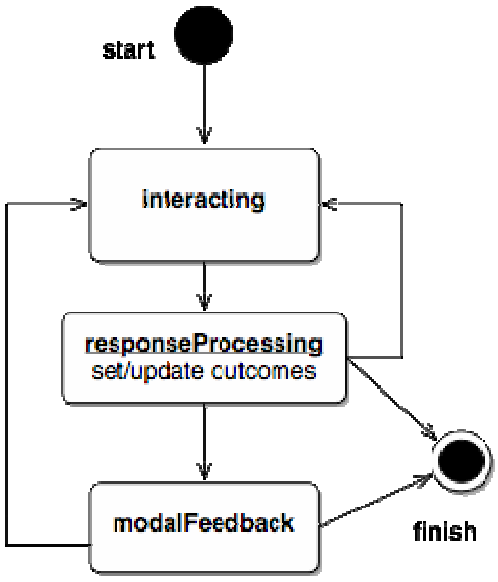


Ilustración 62: Diagrama de estados Feedback

El proceso de la respuesta implica el uso de un sistema de Reglas de respuestas, incluyendo las condiciones de respuesta y la evaluación de las expresiones que implican las variables de la pregunta. Este sistema de entrega de respuestas encajonado supone una barrera a la adopción de la especificación, para aliviar este problema, se pone en práctica un proceso generalizado de respuesta como una característica opcional, para ello el sistema implementa un pequeño número de procesos estándar para la corrección de las preguntas. Estos estándares se describen usando el lenguaje de proceso definido en esta especificación y se distribuyen en plantillas en formato XML. De manera opcional la persona editora de preguntas puede generar plantillas personificadas de corrección de preguntas que posteriormente podrá intercambiar con la comunidad de usuarios que siga este estándar. Los procesos de entrega de respuestas no necesitan soporte especial en ejecución ya que se analiza el archivo correspondiente a la respuesta referenciada de forma directa.

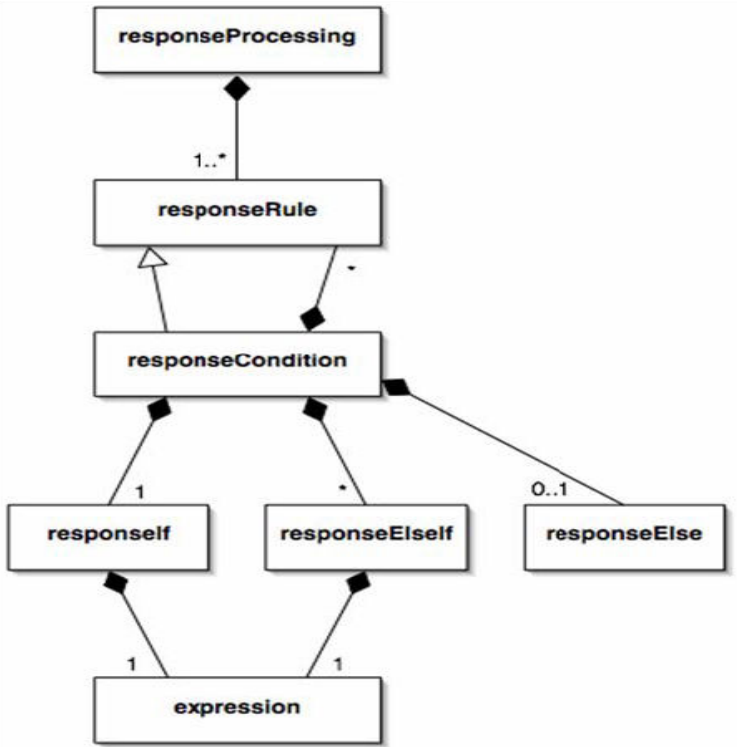


Ilustración 63: Diagrama del Proceso de Respuestas

9 Arquitectura del Sistema

9.1 Justificación de la arquitectura elegida

9.1.1 Elección de capas y componentes

El objetivo principal de la arquitectura es separar, de la forma más limpia posible, las distintas capas de desarrollo, con especial atención a permitir un modelo de desarrollo de código limpio y ordenado y a facilitar posibles acciones futuras de mantenimiento y evolución de las aplicaciones. Otros elementos importantes han sido la facilidad del despliegue, el uso de software libre y el empleo de las mejores tecnologías disponibles en la actualidad.

Para ello se deseaba una arquitectura que permitiese trabajar en capas y que sirviese tanto para las aplicaciones en la Intranet como en Internet, para facilitar su uso tanto desde un ordenador de la facultad como desde la casa de cualquiera de los alumnos. Para lograr esto se eligió el patrón MVC (Modelo-Vista-Controlador) que permite una separación limpia entre las distintas capas de una aplicación. Para la capa de presentación (la vista) se buscaba un framework que nos proporcionase una mayor facilidad en la elaboración de pantallas, mapeo entre los formularios y sus clases en el servidor, la validación, conversión, gestión de errores, traducción a las diversas lenguas autonómicas y, de ser posible, que facilitase también el incluir componentes complejos (menús, árboles, ajax, etc.) de una forma sencilla y –sobre todo- fácil de mantener. Para esta capa se ha elegido Java Server Faces.

En la capa de negocio y persistencia, hemos optado por una solución basada en servicios que trabajaban contra un modelo de dominio limpio. La persistencia de las clases se sustenta en DAO's (Objetos de Acceso a Datos), manteniendo aislada la capa de persistencia de la capa de negocio. Tanto los servicios como los DAO's así como el propio modelo son realmente POJOs (clases simples de Java), con la simplicidad que conllevan y sin dependencias reales con ningún framework concreto. Para realizar esta integración se ha elegido Spring.

Para la capa de persistencia hemos utilizado utilizar una herramienta ya existente, que nos permite realizar el mapeo objeto-relacional de una forma cómoda pero potente, sin tener que implementarlo directamente mediante HSQLDB o MySQL. Esto último conllevaría, por ejemplo, un esfuerzo importante en un caso de cambio de base de datos (como ha ocurrido, ya que tuvimos que pasar de HSQLDB a MySQL), en la gestión de la caché, la utilización de carga perezosa, etc. La herramienta elegida finalmente fue Hibernate.

9.1.2 Beneficios de la arquitectura diseñada

La primera ventaja se deriva de la modularidad del diseño. Cada una de las partes empleadas (JSF para la vista, Spring para la integración e Hibernate para la persistencia) es intercambiable de forma sencilla y limpia por otras soluciones disponibles. Por ejemplo, para la vista se emplea Java-Server Faces, pero nada impide emplear también una aplicación de escritorio mediante Swing o SWT sin tener que tocar ni una sola línea de código de las restantes capas. Es más, nada impediría que se pudiese disponer de una aplicación con una parte de la capa de presentación en JSF y otra parte, para otro tipo de usuarios, en Swing, ambas funcionando a la vez y compartiendo todo el resto del código (lógica de negocio, persistencia, integración, seguridad, etc.).De igual forma, si se desean cambiar elementos de la capa de persistencia empleando otro framework para el mapeo diferente de Hibernate –o sencillamente no utilizar ninguno- tan sólo serían necesarios cambios en esa capa. De la misma manera se podrían sustituir cualquiera de las otras capas. El diseño se ha hecho reduciendo al mínimo posible las dependencias entre ellas.

9.2 J2EE

9.2.1 Introducción a J2EE

J2EE es un estándar que ofrece un modelo distribuido multicapa, componentes reutilizables, un modelo de seguridad unificado, control de transacciones flexibles y soporte para servicios Web a través de protocolos y estándares basados en XML.
La lógica de la aplicación se divide en componentes según su funcionalidad, y los componentes de la aplicación que constituyen la aplicación J2EE se instalan en diferentes máquinas dependiendo de la capa a la que pertenezcan.

- Componentes de la capa del cliente (*client-tier*) que se ejecutan en la máquina del cliente.
- Componentes de la capa Web (*Web-tier*) que se ejecutan en el servidor J2EE.
- Componentes de la capa de Negocios (*Bussiness-tier*) que se ejecutan en el servidor J2EE.
- Software de la capa del sistema de información (*Enterprise information system-tier*) que se ejecuta en el servidor EIS.

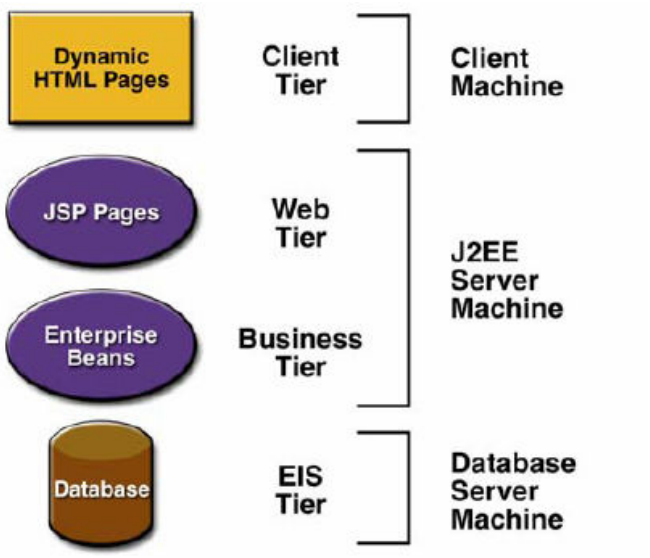


Ilustración 64: Aplicación J2EE

Las aplicaciones J2EE se suelen considerar de tres capas porque se distribuyen en tres lugares: máquina del cliente, máquina del servidor J2EE y la base de datos (*back end*).

9.2.2 Componentes J2EE

Un componente J2EE es una unidad de software funcional auto-contenido que se une a una aplicación J2EE con sus clases y ficheros, y que se comunica con otros componentes. La especificación del estándar J2EE define los siguientes componentes:

- Aplicaciones cliente y Applets son componentes que se ejecutan en el lado del cliente.
- Servlets y Java Server Pages (JSP) son componentes Web que se ejecutan en el servidor.
- Enterprise JavaBeans (EJB) son componentes de negocio que se ejecutan en el servidor.

9.2.3 Aplicaciones Web con tecnologías Java.

Java ofrece las siguientes tecnologías para el desarrollo de aplicaciones Web:

Java Servlet: es una clase Java que procesa peticiones y construye respuestas dinámicamente. Los servlets son muy apropiados para servicios Web y funciones de control.

Java Server Pages (JSP): son documentos basados en texto que se ejecutan como servlets pero que ofrecen un enfoque más natural para la creación de contenido estático. Las JSP se usan para la generación de texto de marcado como HTML o XML.

Java Server Pages Standard Tag Library (JSTL): es una librería de etiquetas que agrupa las funcionalidades más comunes para la creación de JSP.

Java Server Faces (JSF): es un *framework* (patrón de desarrollo) basado en el modelo MVC. Permite que las aplicaciones Web gestionen la complejidad de la interfaz de usuario en el servidor, de modo que el desarrollador se centra en el código de la aplicación.

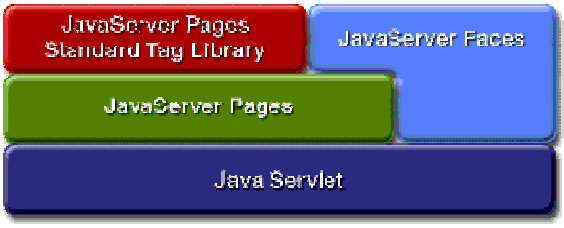


Ilustración 65: Tecnologías Web

Los servlets son la base de todas las aplicaciones Web. Cada tecnología añade un nivel de abstracción que hace que el desarrollo de aplicaciones Web sea más rápido y robusto.

9.2.4 Java Servlets

Definición: es una clase Java que se usa para extender las capacidades de los servidores que albergan aplicaciones a través de peticiones-respuestas (*request - response*).

Usos principales:

- Procesar y almacenar datos enviados desde un formulario HTTP.
- Proveer contenido dinámico (Ej. accediendo a una BD)
- Administrar sesiones para las peticiones HTTP.

Ciclo de vida: cuando una petición se mapea a un servlet, el contenedor en el que se encuentra dicho servlet realiza los siguientes pasos:

- Si no existe una instancia del servlet
    - Se carga la clase del servlet
    - Se crea una instancia de la clase
    - Se inicializa la instancia llamando al método *init*.
- Se invoca al método ***service*****,** pasando los objetos de petición y respuesta.
- Si el contenedor necesita borrar el servlet, se llama al método *destroy*.

Ejemplo de servlet.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet{

 public void doGet(HttpServletRequest request,HttpServletResponse response)
                  throws ServletException, IOException {

    PrintWriter out = response.getWriter();

    out.println("Hello World");
  }
}
```

9.2.5 Java Server Pages (JSP)

Definición: es un documento de texto que contiene dos tipos de texto: datos estáticos (HTML, XML, etc.) y elementos JSP que construyen contenido dinámico.

Ciclo de vida: el servicio de una página JSP se procesa con un servlet. Cuando una petición se mapea a una JSP, el contenedor Web primero comprueba si el servlet de la página es más antiguo. Si esto ocurre, el contenedor traduce la página JSP en una clase servlet y la compila. Este proceso se genera automáticamente.

Ejemplo de página JSP.

```
<%@page contentType="text/html"%>
<html>
        <head>
<title>
Hello JSP
</title>
</head>
         <body>
                <h1>Hello World</h1>
                 The time is <%= new java.util.Date() %>
         </body>
 </html>
```

9.2.6 Java Server Pages Standard Tag Library (JSTL)

Definición: es una librería de etiquetas que reúne la funcionalidad más común para las páginas JSP.

Librerías:

- **Core**: etiquetas básicas para iteraciones, instrucciones condicionales y entrada/salida.
- **XML**: procesamiento de documentos XML.
- **Internacionalización**: formato I18N.
- **SQL**: etiquetas para acceso a base de datos.

Ejemplo de página JSP con JSTL.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
        <body>
                <c:out value="Hello world!"/>
        </body>
</html>
```

9.2.7 Java Server Faces

Java Server Faces es el estándar presentado por Sun para la capa de presentación Web. Forma parte de la especificación J2EE 5 -que deberán cumplir todos los servidores de aplicaciones- y se erige como una evolución natural de los frameworks actuales hacia un sistema de componentes. Es un estándar sencillo que aporta los componentes básicos de las páginas Web además de permitir crear componentes más complejos (menús, pestañas, árboles, etc.). Ya hay disponibles diferentes implementaciones de la especificación, tanto comerciales como de código abierto, así como librerías de componentes adicionales que amplían la funcionalidad de esos componentes iniciales.

JSF ha sido acogida por la comunidad como “el framework que hacía falta”. Muchos de los proyectos de código abierto y las compañías con más influencia lo han identificado como el framework de presentación Web del futuro. JBoss ha integrado directamente JSF con EJB3 mediante el proyecto Seam (abanderado además por Gavin King, el líder del equipo de desarrollo Hibernate). IBM lo ha incorporado como mecanismo de presentación estándar para su entorno, desarrollando no sólo el soporte completo en su IDE, sino nuevos componentes. Oracle es otra de las compañías que más ha apostado por esta tecnología, ofreciendo la que posiblemente sea la mayor oferta de componentes propios.

Dentro de los “pesos pesados” en Java, nombres que han guiado a la industria en el pasado como Matt Raible, Rick Hightower, David Geary, el mencionado Gavin King y por supuesto el propio creador de Struts y ahora arquitecto de JSF, Craig McClanahan.

Otra de las grandes características de JSF que ha producido su rápida adaptación y su uso masivo ha sido su total compatibilidad con el resto de tecnologías Java que están disponibles, así como su total independencia respecto a ellas en su uso, ya que si nosotros queremos utilizar JSF en una aplicación existente, sólo tenemos que modificar la parte Web de la misma sin tener que cambiar nada del código interno.

9.2.7.1 Aplicaciones Web con Java Server Faces

Una aplicación típica con JSF suele contener los siguientes componentes, todos ellos típicos de cualquier tecnología Java para el diseño de páginas Web:

- Componentes JavaBean que contienen la funcionalidad y datos de la aplicación.
- *Listeners* para eventos (*event listeners*)
- Páginas (Ej. páginas JSP)
- Clases auxiliares del lado del servidor (Ej. *beans* de acceso a base de datos)

Además, las aplicaciones con tecnología JSF también tienen:

- Librería de etiquetas para la representación de componentes UI.
- Librería de etiquetas para representar manejadores de eventos, validadores, y otras acciones.
- Componentes UI representados como objetos con estado en el servidor.
- *Beans* de soporte (*backing beans*), que definen propiedades y funciones para los componentes UI.
- Validadores, conversores, *listeners* de eventos, y manejadores de eventos.
- Fichero de configuración de recursos de la aplicación (Ej. faces-config.xml).

La librería de etiquetas de componentes evita la necesidad de escribir código directamente en HTML u otro lenguaje de marcado. La librería de etiquetas principal (*core tag library*) facilita el registro de eventos, validadores y otras acciones sobre los componentes.

9.2.7.2 Ejemplo de Aplicación usando Java Server Faces

Para ver un ejemplo de uso de la tecnología Java, dirijase al anexo A y siga el manual LibraryWeb.

9.3 Spring

Spring es un *framework* indicado para desarrollar aplicaciones escritas en Java, es decir, para el desarrollo de aplicaciones J2EE *(Java 2 Enterprise Edition*), incluyendo EJB (*Enterprise JavaBeans*), Servlets y JSP (*Java Server Pages*). Spring logra combinar dichas herramientas y otras más en un sólo paquete, proporcionando una estructura más sólida y un mejor soporte para este tipo de aplicaciones. Spring no intenta inventar nada sino integrar las diferentes tecnologías existentes en un único *framework* para el desarrollo más sencillo y eficaz de aplicaciones J2EE portables entre servidores de aplicaciones, dándonos un conjunto de librerías, que el programador elige "a la carta" para facilitar el desarrollo de su aplicación.

Entre sus funcionalidades más potentes está su contenedor de Inversión de Control (Inversión de Control, también llamado Inyección de Dependencias, es una técnica alternativa a las clásicas búsquedas de recursos vía JNDI. Permite configurar las clases en un archivo XML y definir en él las dependencias. De esta forma la aplicación se vuelve muy modular y a la vez no adquiere dependencias con Spring), la introducción de aspectos, plantillas de utilidades para Hibernate, iBatis y JDBC así como la integración con JSF.

A pesar de su abundante funcionalidad Spring esta diseñada para permitir la colaboración en el trabajo entre cada uno de los componentes que contiene y otros externos pero manteniendo la modularidad y la independencia de cada uno de las partes que contienen una aplicación que usa Spring. Esto permite la posibilidad de retirarlo sin prácticamente cambiar líneas de código. Lo único que sería necesario es, lógicamente, añadir la funcionalidad que provee, ya sea con otro framework similar o mediante nuestro código. Además en contra de lo que se podría pensar Spring es un frame lightweight, es decir, un framework ligero, ya que no requiere muchos recursos para su ejecución además de por su reducido tamaño (es un .jar de alrededor de 1MB).

Otras de las características de Spring es su constante evolución, ya que hay multitud de proyectos que se encargan del aumento de la funcionalidad de este framework así como de su compatibilidad con casi la totalidad de las tecnologías más usadas en el desarrollo de aplicaciones. Esto es posible ya que Spring es una aplicación OpenSource, lo que permite a cualquier desarrollador su uso sin coste alguno.

9.3.1 Arquitectura de Spring

Spring es un *framework* modular que cuenta con una arquitectura divida en siete capas o módulos, como se muestra en la Figura 61, lo que nos permite tomar y ocupar únicamente las partes que nos interesen para nuestro proyecto e integrarlas con gran libertad.

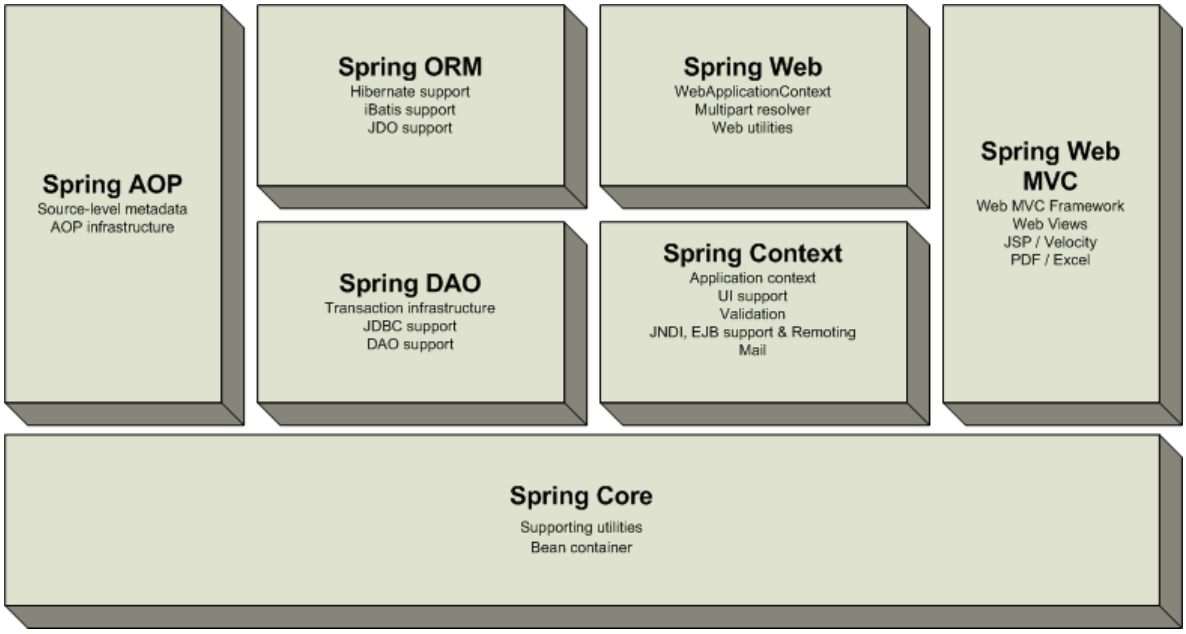


Ilustración 66: Arquitectura de Spring

Dos de los paquetes más importantes de Spring son `org.springframework.beans` y `org.springframework.context`. El código de estos paquetes proporciona la base de las características principales de la Inversión de Control (*IoC*) de Spring.

BeanFactory proporciona un mecanismo avanzado de configuración que permite la administración de beans (objetos) de cualquier naturaleza, usando potencialmente cualquier tipo de almacenamiento. El contexto de la aplicación (*ApplicationContext*) se construye encima de *BeanFactory*, añadiendo mejoras y otras funcionalidades.

Resumiendo, *BeanFactory* proporciona la configuración del *framework* y la funcionalidad básica, mientras que el *ApplicationContext* añade además otras características mejoradas. Cualquier descripción sobre las capacidades y el comportamiento de *BeanFactory* se puedes aplicar al *ApplicationContext*.

Normalmente cuando se construyen aplicaciones basadas en J2EE, la mejor opción es usar el *ApplicationContext* porque ofrece todas las características de *BeanFactory* y además permite un enfoque más declarativo parar usar parte de la funcionalidad, lo que siempre suele ser recomendable.

9.3.1.1 Spring Core

Esta parte es la que provee la funcionalidad esencial del *framework.* Está compuesta por `BeanFactory que` utiliza el patrón de Inversión de Control (*Inversion of Control*) y configura los objetos a través de Inyección de Dependencia (*Dependency Injection*).

9.3.1.1.1 Bean Factory

Es uno de los componentes principales del núcleo de Spring. Es una implementación del patrón *Factory*, pero a diferencia de las demás implementaciones de este patrón, que muchas veces sólo producen un tipo de objeto, `BeanFactory` es de propósito general, ya que puede crear muchos tipos diferentes de *Beans*. Los *Beans* se pueden llamar por su identificador y ellos mismos se encargan de manejar las relaciones entre objetos. Se soportan objetos de dos tipos diferentes:

- *Singleton* – existe únicamente una instancia compartida de un objeto con un nombre particular, que se puede llamar o devolver cada vez que se necesite. Este modo está basado en el patrón de diseño que lleva el mismo nombre.
- *Prototype* – también conocido como *non-singleton*. En este método cada vez que se realiza una llamada se crea un nuevo objeto independiente.

BeanFactory es el contenedor real que instancia, configura y administra los beans. Estos beans normalmente colaboran unos con otros, y los por tanto tienen dependencias entre ellos. Estas dependencias se reflejan en los datos de configuración usados por *BeanFactory*.

La configuración de un *BeanFactory,* en el caso más básico, consiste de definiciones de uno o más beans que deberá administrar. En una factoría definida a través de XML, los beans se declaran de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
 "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

       <bean id="GradingService"
              class="es.eucm.assessment.services.GradingServiceImpl" />

       <bean id="ItemSession"
              class="es.eucm.assessment.session.ItemSessionImpl"
              singleton="false">
                     <property name="gradingService">
                            <ref bean="GradingService"/>
                     </property>
      </bean>

                     . . .

       <bean id="Item"
             class="es.eucm.assessment.pojo.ItemImpl" />

</beans>
```

Las definiciones de los beans se representan a través de objetos *BeanDefinition* que contienen (entre otra información) los siguientes detalles:

- Identificador del bean (*id*) – cada bean tiene uno o más identificadores. Éstos deben ser únicos dentro del *BeanFactory/ApplicationContext* en el que esté almacenado. Un bean tendrá casi siempre un único nombre, pero en el caso de tener más de un identificador, todos los demás identificadores se pueden considerar alias.
- Nombre de la clase (*class*) – clase que implementa la definición del bean.
- Elementos de configuración del comportamiento del bean – como por ejemplo *prototype* o *singleton,* métodos de inicialización y destrucción, etc.
- Argumentos del constructor o propiedades del bean – como por ejemplo el límite de un pool de conexiones de una base de datos.
- Otros beans necesarios para este bean (colaboradores) – también se llaman dependencias.

9.3.1.1.2 Inversion of Control

Una de las funcionalidades más importantes de Spring es el uso del patrón de Inversión de Control (*IoC, Inversion of Control)* `utilizado por el BeanFactory`. Esta parte se encarga de separar del código de la aplicación que se está desarrollando, los aspectos de configuración y las especificaciones de dependencia del *framework*. Todo esto se realiza configurando los objetos a través de Inyección de Dependencia o *Dependency Injection,* que se explicará más adelante.

Una forma sencilla de explicar el concepto de *IoC* es el *Principio de Hollywood: No nos llames, no1sotros te llamaremos*. Traduciendo este principio a nuestro contexto, en lugar de que el código de la aplicación llame a una clase de una librería, un *framework* que utiliza *IoC* llama al código. Por esta razón se llama *Inversión*, ya que invierte la acción de llamada a alguna librería externa.

9.3.1.1.3 Dependency Injection

Con este principio, en lugar de que el código de la aplicación utilice el API del *framework* para resolver dependencias como parámetros de configuración u objetos colaborativos, las clases de la aplicación exponen o muestran sus dependencias a través de métodos o constructores que el *framework* puede llamar con el valor apropiado en tiempo de ejecución, basado en la configuración.

El principio básico es que los beans definen sus dependencias (es decir, los otros objetos con los que trabajan) sólo a través de argumentos para métodos constructores, argumentos para una factoría o propiedades que se inicializan una vez el objeto se ha creado o devuelto a través de una factoría. Por lo tanto, el trabajo del contenedor es el de *inyectar* estas dependencias cuando se crea el bean. Esto es fundamentalmente lo inverso (de aquí el nombre) del bean, instanciando o localizando sus dependencias por si mismo usando clases o constructores directamente.

Se ve claramente que el uso de *IoC* proporciona código mucho más limpio y además se logra un alto grado de desacoplamiento entre los beans porque los éstos no buscan sus dependencias directamente.

La inversión de control / Inyección de dependencias tiene dos variantes principales:

- Inyección de dependencias basada en métodos *set* (*setter-based*) – se realiza llamando a los métodos *set* de los beans después de invocar a un constructor sin argumentos o a un método sin argumentos de una factoría estática para instanciar al bean. Spring generalmente recomienda el uso de este tipo de inyección de dependencias porque un constructor con muchos argumentos se puede volver poco manejable, especialmente cuando algunas propiedades son opcionales.
- Inyección de dependencias basada en constructores (*constructor-based*) – se realiza invocando a un constructor con un número de argumentos, cada uno representando a un colaborador o una propiedad. Aunque Spring generalmente recomienda la otra aproximación para la inyección de dependencias, también se soporta completamente la inyección basada en constructores para permitir el uso de beans ya predefinidas que puedan proporcionar constructores con argumentos pero métodos *set*. Además, esta aproximación se puede usar como mecanismo para asegurar que beans sencillos no se construyan en un estado invalido.

La resolución de dependencias de los beans se desarrolla generalmente así:

- Se crea e inicializa el *BeanFactory* con la configuración que describe a todos los beans. Normalmente se usa una variante para el *BeanFactory* o *ApplicationContext* que soporta formato XML para los ficheros de configuración.
- Cada bean tiene dependencias expresadas en forma de propiedades. Estas dependencias serán proporcionadas al bean en el momento de creación.
- Cada propiedad o argumento del constructor es una definición real del valor a establecer o una referencia a otro bean en el *BeanFactory*.
- Es importante darse cuenta que Spring valida la configuración para cada bean del *BeanFactory* cuando éste se crea, incluyendo la validación de las propiedades que sean referencias a otros beans válidos. Sin embargo, las propiedades del bean no se asignan realmente hasta que se crea. Para beans que son *singleton*, la creación se realiza cuando el *BeanFactory* es creado, pero en cualquier otro caso el bean se crea cuando se necesita.
- Generalmente se puede suponer que Spring realizará bien las cosas. Se resolverán los problemas de configuración, referencias a beans no existentes y dependencias circulares. Las propiedades se asignan y se resuelven las dependencias tan tarde como sea posible, que suele ser cuando el bean es creado.

A continuación se muestra un ejemplo:

Primero se muestra una parte del fichero de configuración en XML en el que se definen algunos beans. A continuación se muestra el código del bean, con el método *set* correspondiente para realizar la inyección de dependencias.

```
<bean id="GradingService"
       class="es.eucm.assessment.services.GradingServiceImpl" />

<bean id="ItemSession"
       class="es.eucm.assessment.session.ItemSessionImpl"
       singleton="false">
              <property name="gradingService">
                     <ref bean="GradingService"/>
              </property>
</bean>
```

```
public class ItemSessionImpl implements ItemSession {

       private GradingService gradingService;

       public void setGradingService(GradingService service){
              gradingService = service;
       }

              . . .

}
```

Como se puede ver, el método *set* de ItemSessionImpl se ha declarado para que coincida con la propiedad definida en el fichero de configuración.

9.3.1.2 Spring Context

Mientras que el paquete *beans* proporciona la funcionalidad básica para administrar y manipular beans, normalmente de un modo programático, Spring *context* añade el *ApplicationContext* que mejora la funcionalidad del *BeanFactory*.

En sí, *Spring Context* es un archivo de configuración que provee de información contextual al *framework* general. Además provee servicios *enterprise* como JNDI, EJB, e-mail y validación.

Application Context

La base para el paquete *context* es la interfaz *ApplicationContext* que se encuentra en `org.springframework.context`. Derivando de la interfaz de *BeanFactory,* Spring Context proporciona toda la funcionalidad de ese interfaz y además añade información de la aplicación que se puede utilizar por todos los componentes, proporcionando además lo siguiente:

- Localización y reconocimiento automático de las definiciones de los *Beans*
- Carga de múltiples contextos jerarquizados, permitiendo que cada uno se enfoque en una determinada capa.
- Herencia de contextos
- Jerarquía de mensajes y soporte para internacionalización (i18n)
- Acceso a recursos

- Propagación de eventos, para permitir que los objetos de la aplicación publicar y opcionalmente registrarse para ser notificados de los eventos.

Como el *ApplicationContext* incluye toda la funcionalidad de *BeanFactory*, se recomiendo normalmente el uso del primero, excepto para algunas situaciones en las que el consumo de memoria sea crítico (es decir, en un *Applet*).

9.3.1.3 Spring AOP

La programación orientada a aspectos (*Aspect-oriented programming, AOP*) es una técnica que permite a los programadores modularizar ya sea las preocupaciones *crosscutting*, o el comportamiento de las divisiones de responsabilidad, como:

- Persistencia
- Manejo de transacciones
- Seguridad
- *Logging*
- *Debugging*

AOP es un enfoque diferente y un poco más complicado de acostumbrarse en comparación con la programación orientada a objetos, pero es un complemento no un rival o una causa de conflicto.

Spring AOP es portable entre servidores de aplicación y funciona tanto en servidores Web como en contenedores EJB.

Spring AOP soporta las siguientes funcionalidades:

- Intercepción – se puede insertar comportamiento personalizado antes o después de invocar a un método en cualquier clase o interfaz.
- Introducción – especificando que un *advice* (acción tomada en un punto particular durante la ejecución de un programa) debe causar que un objeto implemente interfaces adicionales.
- *Pointcuts* dinámicos y estáticos – para especificar los puntos en la ejecución del programa donde debe de haber intercepción.

9.3.1.4 Spring ORM

En lugar de proporcionar su propio módulo ORM (*Object-Relational Mapping*), para los usuarios que no se sientan confiados en utilizar simplemente JDBC, Spring propone un módulo que soporta los *frameworks* ORM más populares del mercado, entre ellos:

- Hibernate (2.1 y 3.0) – es una herramienta de mapeo O/R *OpenSource* muy popular que utiliza su propio lenguaje de *query* llamada HQL.
- iBATIS SQL Maps (1.3 y 2.0) – es una solución sencilla pero poderosa para hacer externas las declaraciones de SQL en archivos XML.
- Apache OJB (*ObJectRelationalBridge*, 1.0) – es una plataforma de mapeo O/R con múltiples APIs para clientes.
- Otros como JDO (*Java Data Objects*, 1.0 y 2.0) y Oracle TopLink.

Todo esto se puede utilizar en conjunto con las transacciones estándar del *framework*. Spring e Hibernate es una combinación muy popular.

Algunas de las ventajas que brinda Spring al combinarse con alguna herramienta ORM son:

- Manejo de sesión – Spring hace de una forma más eficiente, sencilla y segura la forma en que se manejan las sesiones de cualquier herramienta ORM que se quiera utilizar.
- Manejo de recursos – se puede manejar la localización y configuración de los SessionFactories de Hibernate o las fuentes de datos de JDBC.
- Manejo de transacciones integrado – se puede utilizar una plantilla (*template*) de Spring para las diferentes transacciones ORM.
- Envolver excepciones – con esta opción se pueden envolver todas las excepciones para evitar las engorrosas declaraciones y los bloques *try–catch* en cada segmento de código.
- Evita limitarse a un sólo producto – si se desea migrar o actualizar a otra versión de un ORM distinto o del mismo, Spring trata de no crear dependencias entre la herramienta ORM, el mismo Spring y el código de la aplicación, para que cuando sea necesario migrar a un nuevo ORM no sea necesario realizar tantos cambios.
- Facilidad de prueba – Spring trata de crear pequeños pedazos que se puedan aislar y probar por separado, ya sean sesiones o un *datasource*.

9.3.1.5 Spring DAO (DAO y JDBC)

El patrón DAO (*Data Access Object*) es uno de los patrones más importantes y usados en aplicaciones J2EE, y la arquitectura de acceso a los datos de Spring provee un buen soporte para este patrón.

Existen dos opciones para llevar a cabo el acceso, conexión y manejo de bases de datos: utilizar alguna herramienta ORM o utilizar la plantilla de JDBC (*Java Database Connectivity*) que brinda Spring. La elección de una de estas dos herramientas es totalmente libre y el desarrollador se debe basar en la complejidad de la aplicación para elegir entre ellas. Si es una aplicación sencilla en la que únicamente una clase realizará la conexión con la base de datos, entonces la mejor opción sería Spring JDBC. En caso contrario, cuando se requiera un mayor soporte y la aplicación sea más robusta se recomienda utilizar una herramienta ORM.

El uso de JDBC muchas veces lleva a repetir el mismo código en distintos lugares como por ejemplo al crear la conexión, buscar información, procesar los resultados y cerrar la conexión. El uso de las dos tecnologías mencionadas anteriormente nos ayuda a mantener simple este código y evitar que sea tan repetitivo, además minimiza los errores al intentar cerrar la conexión con algunas bases de datos.

9.3.1.6 Spring Web

El módulo Web de Spring se encuentra en la parte superior del módulo de contexto, y provee el contexto para las aplicaciones Web. Este módulo proporciona el soporte necesario para la integración con el *framework Struts* de Yakarta.

Este módulo también se encarga de diversas operaciones Web como por ejemplo ejecutar las peticiones multi-parte que puedan ocurrir al realizar cargas de archivos y la relación de los parámetros de las peticiones con los objetos correspondientes (*domain objetcs o business objects*).

9.3.1.7 Spring Web MVC

Spring brinda un patrón MVC (*Model View Controller*) para aplicaciones Web bastante flexible y altamente configurable, pero esta flexibilidad no le quita sencillez, ya que se pueden desarrollar aplicaciones sencillas sin tener que configurar muchas opciones.

Para esto se puede utilizar muchas tecnologías ya que Spring ofrece soporte para JSP, *Struts* y Velocity, entre otros.

El módulo Web MVC de Spring presenta algunas similitudes con otros *framework* que existen en el mercado, pero estas características lo vuelven único:

- Spring hace una clara división entre controladores, modelos de *JavaBeans* y vistas.
- El MVC de Spring esta basado en interfaces y es bastante flexible.
- Provee interceptores (*interceptors*) al igual que controladores.
- Spring no obliga a utilizar JSPs como única tecnología para la vista (*View*).
- Los controladores son configurados de la misma manera que los demás objetos en Spring, a través de IoC.
- Las capas Web (*Web tiers)* son más sencillas de probar que en otros *frameworks*.

Para intentar comprender cada parte de la arquitectura del Web MVC de Spring, se presenta en la figura 62 el ciclo de vida de una petición (*request*):

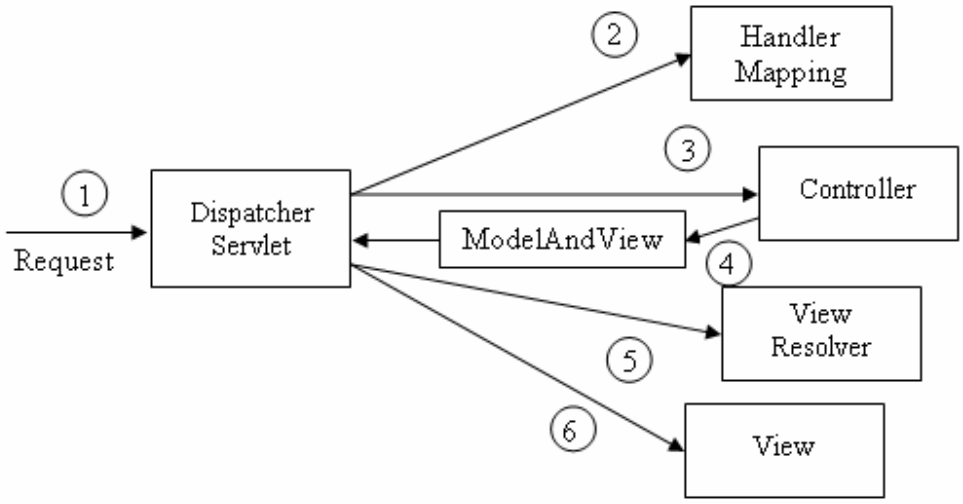


Ilustración 67: Ciclo de vida de una petición Spring

1. El navegador manda una petición y lo recibe un `DispatcherServlet`
2. Se debe escoger que controlador (`Controller`) manejará la petición. Para esto el `HandlerMapping` mapea los diferentes patrones de URL hacia los controladores, y se le devuelve al `DispatcherServlet` el `controlador` elegido.
3. El `controlador` elegido toma la petición y ejecuta la tarea.
4. El `controlador` devuelve un `ModelAndView` al `DispatcherServlet`.
5. Si el `ModelAndView` contiene un nombre lógico de un `View` se tiene que utilizar un `ViewResolver` para buscar ese objeto `View` que representará la petición modificada.
6. Finalmente el `DispatcherServlet` despacha la petición al `View`.

Spring cuenta con una gran cantidad de controladores de los cuales se puede elegir dependiendo de la tarea, entre los más populares se encuentra: *SimpleController* y *AbstractController*.

9.3.2 Administración de Transacciones con Spring

Spring proporciona una abstracción consistente para la administración de transacciones, siendo esta abstracción es uno de los mayores beneficios de Spring. Además también aporta las siguientes ventajas:

- Proporciona un modelo de programación consistente para diferentes APIs como JTA, JDBC o Hibernate.
- Proporciona una gestión de transacciones más sencilla y fácil de usar que la mayoría de las APIs para transacciones.
- Da soporte para la administración declarativa de transacciones.

Los desarrolladores tienen principalmente dos opciones para realizar esta administración: transacciones globales o transacciones locales.

Las transacciones globales se administran a través de un servidor de aplicaciones usando JTA (*Java Transaction API*), lo que permite la posibilidad de gestionar múltiples recursos transaccionales (aunque la mayoría de las aplicaciones usa un único recurso). El uso de JTA para el código es la principal desventaja de esta opción debido a su complejo e incomodo modelo de excepciones.

Las transacciones locales son más fáciles de usar pero también tienen desventajas significativas: tienden a invadir el modelo de programación. Por ejemplo, código que administra transacciones usando JDBC no puede ejecutarse dentro de una transacción JTA.

Spring resuelve estos problemas permitiendo a los desarrolladores el uso de un modelo consistente en cualquier entorno. También proporciona tanto administración programática como declarativa. Esta última es la más recomendaba en la mayoría de los casos gracias a que la administración de las transacciones se puede hacer con poco código y por lo tanto no depende de ningún API.

9.3.2.1 Administración Declarativa de Transacciones

Spring ofrece la administración declarativa de transacciones a través de la Programación Orientada a Aspectos (*AOP, Aspect Oriented Programming*). Este tipo de transacciones es el más usado porque es la opción que menos impacto genera en el código de la aplicación. Las características principales son:

- Spring permite especificar el comportamiento de la transacción para cada método a través de *AOP*.
- Spring trabaja en cualquier entorno. Puede trabajar con JDBC, JDO o Hibernate, simplemente cambiando la configuración.
- Spring ofrece reglas declarativas de *rollback*.
- Spring no soporta propagación de contextos en las transacciones a través de llamadas remotas. Se recomienda usar EJB (*Enterprise JavaBeans*) si se desea usar esta característica.

El concepto de reglas de *rollback* es importante: permite especificar que tipo de excepciones deberían causar un *rollback* automático. Esto se especifica declarativamente en la configuración, no en código Java. Esto tiene la principal ventaja de que los objetos de negocio no dependen de la infraestructura de las transacciones.

La forma habitual de configurar un proxy de transacciones en Spring es a través del uso de `TransactionProxyFactoryBean`. Esta factoría es simplemente una versión especializada de la factoría genérica `ProxyFactoryBean` que crea un proxy para envolver el objeto para el que se van a gestionar las transacciones.

Para usar el bean `TransactionProxyFactoryBean`, primero se necesita especificar el objeto que se va a envolver con el proxy de transacciones a través del atributo `target`. Este objeto es normalmente un bean POJO (*Plain Old Java Object*). También se debe definir una referencia a `HibernateTransactionManager`. Finalmente, se deben especificar los atributos de la transacción, `transactionAttributes`. Estos atributos

contienen la definición de la semántica de las transacciones así como los métodos donde se aplican. Consideremos el siguiente ejemplo del proyecto QTI:

```
<bean id="ValueDAOService"
        class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
        <property name="transactionManager" ref="transactionManager" />
        <property name="target" ref="ValueDAOTarget" />
        <property name="transactionAttributes">
                <props>
                        <prop key="*">PROPAGATION_REQUIRED</prop>
                </props>
        </property>
</bean>

<bean id="ValueDAOTarget"
        class="es.eucm.assessment.dao.pojo.outcomes.ValueDAOImpl">
        <property name="sessionFactory" ref="mySessionFactory" />
</bean>

<bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="mySessionFactory"/>
</bean>

<bean id="mySessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource" ref="myDataSource"/>
        <property name="mappingResources"> [...] </property>
        <property name="hibernateProperties"> [...] </property>
</bean>
```

Los atributos de la transacción se definen a través de un par consistente en el nombre del método y el valor. El valor es una cadena de texto con el siguiente formato:

```
PROPAGATION_NAME,ISOLATION_NAME,readOnly,timeout_NNNN,+Exception1,-Exception2
```

PROPAGATION_NAME: normalmente todo el código se ejecuta dentro del ámbito de una misma transacción. Sin embargo hay varias opciones para especificar el comportamiento si un método transaccional se ejecuta cuando otra transacción ya existía.

ISOLATION_NAME: grado de aislamiento de esta transacción respecto a otras transacciones.

timeout_NNNN: específica cuanto tiempo se debe esperar a la transacción antes de hacer *time out*, provocando automáticamente *rollback*.
+Exception1: especifica que esta excepción fuerza un *commit* cuando se lanza.
–Exception2: especifica que esta excepción fuerza un rollback cuando se lanza.

A continuación se muestra un diagrama que representa el modo en el que Spring ensambla todos los beans de una parte del proyecto QTI según la configuración de la página anterior.

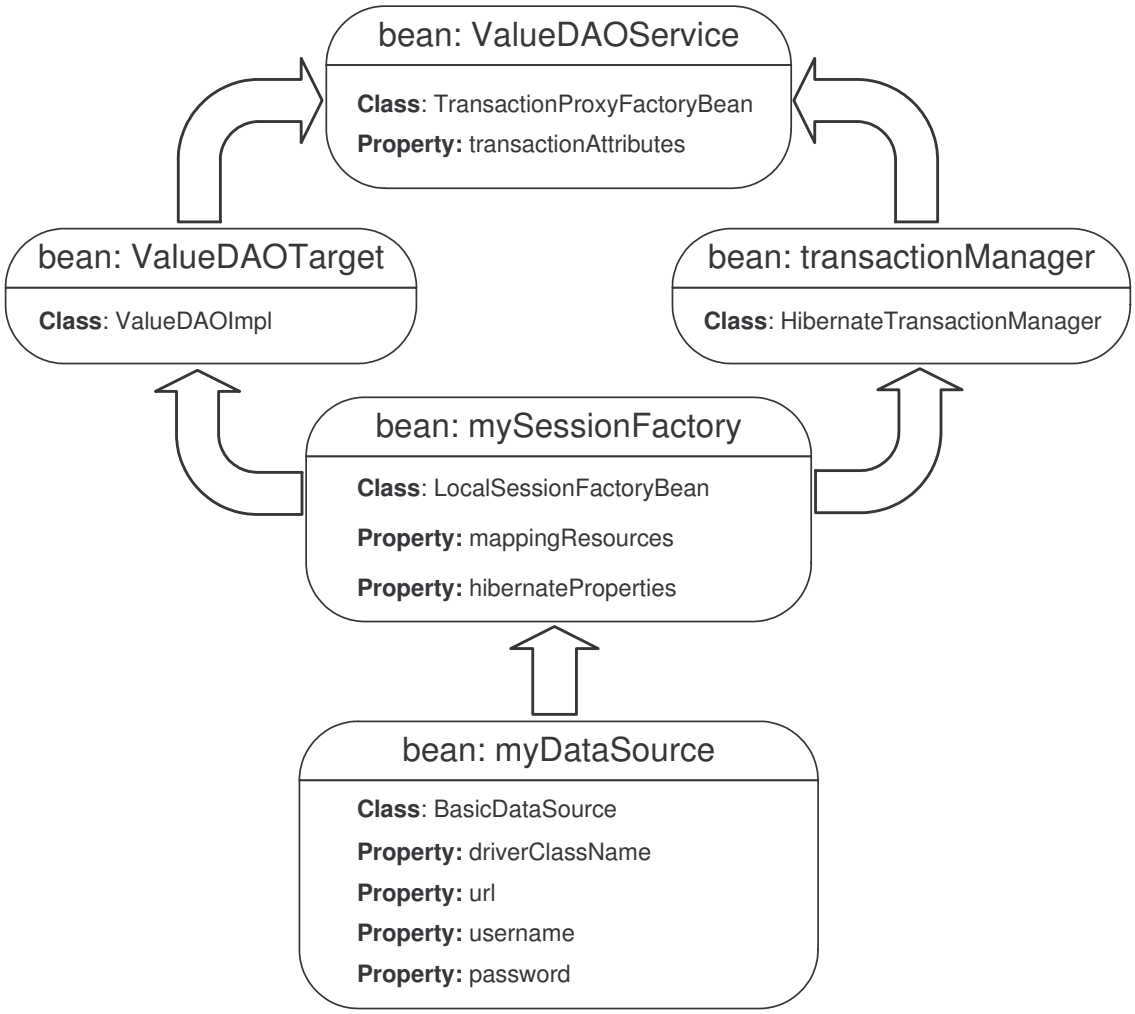


Ilustración 68: Generación de los beans basado en la configuración de Spring

9.3.2.2 Administración Programática de Transacciones

Spring ofrece dos mecanismos para la administración programática de transacciones:

- Usando `TransactionTemplate`: se adopta la misma aproximación que con otras plantillas de Spring.
- Usando `PlatformTransactionManager`: similar al API de JTA.

9.3.2.3 Administración Declarativa vs. Programática

La elección del tipo de gestión de transacciones dependerá de la arquitectura de la aplicación. La administración de transacciones programática es una buena opción si se tiene un número pequeño de operaciones transaccionales. Por el contrario, si la aplicación tiene muchas operaciones con transacciones, la administración declarativa es la adecuada porque mantiene la gestión de las transacciones fuera de la lógica de negocios y es fácilmente configurable en Spring.

9.3.3 Ejemplo de uso de Spring

Para ver un ejemplo de uso de Spring dirijase al anexo A y siga el manual Spring-Hibernate.

9.4 Hibernate

Hibernate es un motor de persistencia de código abierto. Permite mapear un modelo de clases a un modelo relacional sin imponer ningún tipo de restricción en ambos diseños. Cuenta con una amplia documentación, tanto a nivel de libros publicados como disponible gratuitamente en su Web. A nivel comercial está respaldado por JBoss, que proporciona servicios de soporte, consultoría y formación en el mismo.

Actualmente es el rey indiscutible de la persistencia. Desde su versión 1.0, el motor no ha parado de evolucionar, incorporando todas las nuevas ideas que se iban incorporando en este campo.

Hoy en día, en su versión 3.1, ya soporta el estándar EJB 3 (su autor es uno de los principales integrantes del JCP que está definiendo esta especificación) por lo que ya se puede elegir desarrollar aplicaciones empleando EJB 3 -que correrán en cualquier contenedor de EJB's que soporte J2EE 5- o aplicaciones independientes. Esto no solo asegura la inversión de tiempo en Hibernate de cara al futuro, sino que, independientemente de su utilidad , nos hace totalmente independientes del mismo.

9.4.1 Mapeador Objeto-Relacional

La programación orientada a objetos y la base de datos relacional corresponde a dos paradigmas muy diferentes. La programación orientada a objetos trata de objetos, sus atributos y asociaciones unos con otros, mientras que la base de datos corresponde a modelos relacionales.

Cuando se desarrolla una aplicación orientada a objetos con base de datos relacional, la manera tradicional de acceder sería a través de una conexión JDBC directa a la base de datos mediante la ejecución de sentencias SQL. Esto sería útil y efectivo cuando no haya gran número de clases de negocio, ya que el mantenimiento del código esta muy relacionado al modelo de datos relacional. Un mínimo cambio en el modelo de datos implica revisión y posibles modificaciones en el código de la aplicación.

Una implementación más avanzada sería la utilización de unas clases de acceso de datos (DAO). Nuestra capa de negocio se comunicaría con la capa DAO y esta realizaría las operaciones sobre la base de datos. Pero sigue habiendo problemas de mantenimiento y portabilidad.

Un Mapeador Objeto-Relacional es una capa de persistencia que separa el código de la aplicación de la realización de nuestras sentencias SQL contra la Base de Datos. Sus funciones van desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.

9.4.2 Características

Hibernate es la herramienta de mapeo objeto-relacional de código abierto más avanzado y maduro que hay actualmente. Te permite diseñar objetos persistentes de una manera muy rápida y optimizada.

La representación de alto nivel para la arquitectura de Hibernate es mostrada en la figura 60. Hibernate es la capa intermediaria entre la aplicación y la Base de Datos y así proporcionar a la aplicación servicios (y objetos) persistentes.

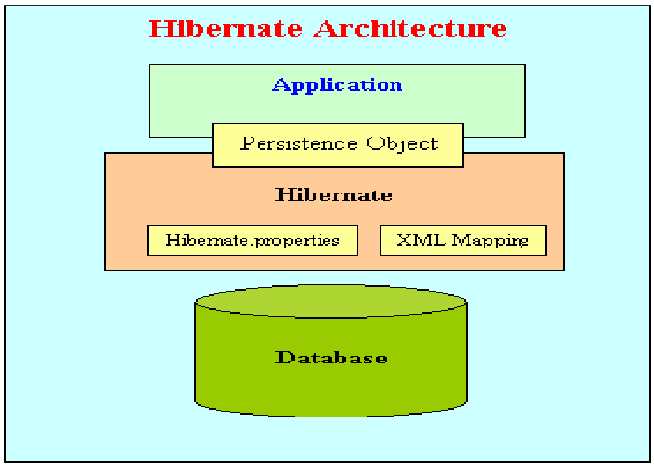


Ilustración 69: Arquitectura de Hibernate

La arquitectura de Hibernate posee tres grandes componentes:

- Gestión de la Conexión: proporciona una gestión eficiente de las conexiones a la base de datos. La aplicación se comunicará con Hibernate únicamente mediante un objeto llamado Sesión e Hibernate se encargará de crear y comunicarse directamente con la base de datos.

- Gestión de Transacciones: permite que el usuario ejecute una o más transacciones en el mismo tiempo.

- Mapeo Objeto-Relacional: transforma o mapea la representación de los datos de un modelo de objetos a un modelo de datos relacional. Este componente de Hibernate consulta, inserta, actualiza y elimina los registros de las tablas de la BBDD. Cuando la aplicación llama al método Session.save() con el objeto deseado, Hibernate lee el estado de los atributos del objeto y ejecuta las instrucciones o consultas necesarias.

Hibernate soporta la mayoría de los sistemas de bases de datos SQL. El *Hibernate Query Language*, diseñado como una extensión mínima, orientada a objetos, de SQL, proporciona un puente elegante entre los mundos objeto y relacional. Hibernate ofrece facilidades para recuperación y actualización de datos, control de transacciones, repositorios de conexiones a bases de datos, consultas programáticas y declarativas, y un control de relaciones de entidades declarativas.

9.4.2.1 Utilidades de Hibernate

Hibernate mapea objetos java a tablas de la Base de Datos, permitiendo de una manera sencilla la persistencia, actualización, consultas y eliminación de los datos almacenados en la BBDD. La manera de cómo mapear los objetos java es definida en un archivo xml, que ha de ser validado con la DTD de Hibernate.
Un ejemplo de mapeo es el siguiente:

Definimos una clase sencilla Persona, con sus atributos y métodos correspondientes.

```
Public class Persona{
        private String nombre;
        private String apellidos;
        private String email;
        private long id;

        public void setNombre(String n){nombre = n;}
        public void setApellidos(String a){apellidos = a ;}
        public void setEmail(String e){email = e;}
        private void setId(Long i){id = i; }
        public String getNombre(){return nombre;}
        public String getApellidos(){return apellidos;}
        public String getEmail(){return email;}
        public Long getId(){return idl;}
}
```

A continuación construimos el archivo xml donde es definido el mapeo de la clase Persona a una tabla PERSONA de la BBDD.

```
<hibernate-mapping>
<class name="Persona" table="PERSONA">
        <id name="id" type="long" column="ID">
        <generator class="assigned"/>
        </id>
        <property name="nombre">
        <column name="NOMBRE">
        </property>
        <property name="apellidos">
        <column name="APELLIDOS">
        </property>
        <property name="email">
        <column name="EMAIL">
        </property>
</class>
</hibernate-mapping>
```

Hibernate a partir de este archivo construye una tabla en la Base de Datos, asignando por cada atributo de la clase Persona una columna diferente. El atributo Id asignará un identificador único con respecto a los otros objetos persistentes. Este atributo corresponderá con la clave principal de la tabla en la BBDD. Utilizando identificadores de objetos tanto a nivel de código como en BBDD simplificamos mucho la complejidad de nuestra aplicación y podemos programar partes de la misma como código genérico.

Para construir las relaciones de la base de datos también se le ha de indicar a Hibernate mediante un archivo xml. En este archivo ha de indicarse que tipo de relación se trata (*one-to-one, one-to-many, many-to-many*), las tablas correspondientes y así como también las claves por las que se relacionaran las tablas.

Una vez creada estas tablas y sus relaciones, si deseamos almacenar, recuperar, actualizar información de esta BBDD, el desarrollador ha de interactuar con el motor de Hibernate mediante un objeto especial llamado Sesión (clase *Session*). Este objeto ofrece diversos métodos para que el desarrollador pueda comunicarse con la base de datos. Por ejemplo:

- *save(Objet object)*: realiza una actualización del objeto pasado como parámetro.
- *createQuery(String queryString)*: realiza una consulta.
- *beginTransaction()*: comienza una transacción sobre la BBDD.
- *close()*: cierra el objeto Session.

Hibernate puede soportar tres lenguajes de consultas diferentes para realizar la comunicación desde la aplicación a la base de datos relacional. Estos son:

- *Hibernate Query Language* (HQL): es una extensión orientada a objetos de SQL. Permite acceder a la información de la BBDD de diferentes maneras incluyendo queries orientadas a objeto. Ejemplo:

```
List users = session.find("from Person as p
       where p.fullName = ?", "Ana Sanz", Hibernate.STRING );
```

- *Hibernate Criteria Query API*: construye consultas dinámicas. Ejemplo:

```
Criteria criteria = session.createCriteria(Person.class);
criteria.add (Expression.eq ("fullName", "Ana Sanz"));
criteria.setMaxResults(20);
List users = criteria.list();
```

- *Native SQL*: soporte para consultas creadas en SQL. Ejemplo:

```
List users = session.createSQLQuery("SELECT {user.*} FROM USERS AS
       {user}", "user", UserInfo.class).list();
```

9.4.2.2 XDoclet

XDoclet2 es una herramienta para la generación de código o XML a partir de "doclets", marcas que se incluyen en los comentarios de un programa. De esta forma, a partir de los comentarios de clases, métodos o variables, es posible generar el XML asociado para Hibernate. Esto facilita el desarrollo y, de forma análoga, documenta el código. XDoclet2 no esta solamente limitado a Hibernate, sino que puede generar toda clase de ficheros descriptores basados en XML, tales como EJB o descriptores de desarrollo de servicio Web. En el caso de Hibernate nos permite crear automáticamente los ficheros xml de mapeo mediante la lectura de meta-atributos y tags escritos en Java.

A continuación se explicará los tags más importantes de XDoclet utilizados para la generación de los archivos de mapeo de Hibernate. En el siguiente ejemplo se muestra la clase Usuario y sus atributos que serán mapeados y así obtener la persistencia de los datos deseada.

La clase Usuario es una entidad que posee un identificador, atributos y asociaciones a otras entidades. Primero declaremos el mapeo para la clase Usuario:

```
/**
* @hibernate.class
* table="USUARIOS"
*/
public class Usuario implements Serializable {
...
```

Los tags de XDoclet para Hibernate siempre tiene la sintaxis @hibernate.tagname (opcional) atributos. El tagname esta relacionado a un elemento dentro de las declaraciones de mapeo XML; en el anterior ejemplo, hibernate.class hace referencia al elemento de mapeo <class> del fichero XML. El atributo table es colocado a USUARIOS. Un extracto del fichero de mapeo generado por este tag es el siguiente:

```
<hibernate-mapping>
<class
name="Usuario"
table="USUARIOS">
...
```

Los usuarios son entidades por lo que se necesita un identificador. En el código fuente de la clase todas las propiedades (valores con tipos o atributos asociados a otras entidades) son marcadas por tags XDoclet en los métodos accesores. Para la propiedad id (identificador) añadimos el siguiente tag al método getId():

```
/**
* @hibernate.id
* column="USER_ID"
* unsaved-value="null"
* generator-class="native"
*/
public Long getId() {
return id;
}
```

Los atributos del tag hibernate.id son los mismos atributos para el elemento <id>.
A continuación mostramos los tags para mapear una simple propiedad de la clase, el nombre del usuario:

```
/**
* @hibernate.property
* column="NOMBRE"
* length="16"
* not-null="true"
* unique="true"
* update="false"
*/
public String getNombre() {
return nombre;
}
```

El tag hibernate.property tiene todos los atributos del elemento <property>. Se puede no utilizar este patrón ya que se puede tomar en cuenta los valores por defecto de Hibernate: Si se añade el tag @hibernate.property al método accesor sin ningún atributo, el mapeo sería <property name=”nombre”/>, también se puede utilizar los valores por defecto para todos los otros posibles atributos. Esta técnica permite una mayor facilidad y rapidez de dominio sobre XDoclet.

Otro componente de la clase Usuario es la Dirección:

```
/**
* @hibernate.component
*/
public Direccion getDireccion() {
return direccion;
}
```

Esta vez los valores por defecto de Hibernate son utilizados para la declaración hiberante.component. Además de esta declaración de mapeo de un componente, las propiedades individuales de Dirección son también mapeadas. En el código fuente de Dirección, se añade los tags hibernate.property a los métodos getCalle(), getCodigoPostal(), getCiudad(). La clase Direccion no es mapeada, es decir no se colocan los tags clase para obtener una nueva tabla, solamente es un componente de Usuario y posiblemente de otras entidades y por lo tanto tampoco tiene un identificador asociado. Únicamente los métodos accesores de las propiedades de los componentes serán marcados mediante tags.

A continuación, terminamos la declaración de mapeo de Usuario con tags para mapear las asociaciones entre distintas entidades.

Mapeo de asociaciones entre entidades

El mapeo de asociaciones entre distintas entidades con XDoclet es básicamente la misma que se utiliza para las propiedades con valores con tipo. Se añaden los tags XDoclet a los métodos accesores. Por ejemplo, la asociación entre Usuario a ítem es la siguiente:

```
/**
* @hibernate.set
* inverse="true"
* lazy="true"
* cascade="save-update"
* @hibernate.collection-key
* column="SELLER_ID"
* @hibernate.collection-one-to-many
* class="Item"
*/
public Set getItems() {
return items;
}
```

Lo primero que se diferencia de la propiedad con valores tipados es el número de tags que se necesita para el mapeo. Se mapea el lado “many” de una asociación one-to-many; por lo tanto el tipo es de clase colección. Los atributos para hibernate.set son los mismos de siempre: inversa para el aspecto bi-direccional y por supuesto carga perezosa. Los otros dos tags están también relacionados a los elementos conocidos XML de Hibernate, <key> y <one-to-many>. Se nombra la columna de la foreign key en la tabla de ítem como SELLER_ID (USUARIO_ID podría ser mas obvio pero menos expresivo) y que se ha de nombrar explícitamente la clase de las entidades referenciadas por el objeto Set.

También se ha de mapear el otro lado de esta asociación. En la clase ítem, se mapea el seller:

```
/**
* @hibernate.many-to-one
* column="SELLER_ID"
* cascade="none"
* not-null="true"
*/

public Usuario getSeller() {
return seller;
}
```

Para el lado “one” de la asociación se puede omitir la clase de la entidad referenciada, esta implícito por el tipo de la propiedad. Ahora tenemos los dos lados de la asociación mapeados y podemos por tanto generar automáticamente los ficheros de mapeo XML.

9.4.3 Ejemplo de uso de Hibernate

Para ver un ejemplo de uso de Hibernate dirijase al anexo A y siga el manual Spring-Hibernate.

9.5 Ant y Maven

9.5.1 Introducción

Antes de comparar Ant con Maven, hay que decir que mientras que Ant es una herramienta de construcción, Maven es una herramienta de gestión de código fuente y que, por tanto, abarca una problemática mucho mayor. Es por eso que Ant, siendo una herramienta de construcción de proyectos, parece tan pobre al lado de Maven.

En general, compensa mucho más Maven que Ant, salvo en algún caso excepcional. Esto es debido a que Ant está mucho más centrado en la construcción del software, mientras que Maven está enfocado hacia el trabajo en equipo. Ant puede compensar en aquellos casos en los que se quiera tener control absoluto del proceso de construcción (y, aún así, se puede integrar una tarea Ant como si fuera un goal Maven).

Ant permite hilar mucho más fino a la hora de hacer el despliegue de la aplicación, pero a costa de ficheros de construcción muchísimo más grandes. Maven, por el contrario, simplifica enormemente el proceso de construcción y despliegue a cambio de seguir una estandarización en el modo de desarrollo. No hay nada que no se puede hacer con Maven que no se pueda hacer con Ant y viceversa, pero en el caso de Ant lleva mucho más trabajo por delante.

9.5.2 Ant

9.5.2.1 Introducción

Ant es una herramienta de código abierto para ensamblar aplicaciones y, de forma general, realizar cualquier operación repetitiva del desarrollo que se pueda mecanizar mediante un script. Hoy por hoy es la herramienta J2EE de construcción más usada y cualquier IDE que se precie se integra nativamente con este programa. Las ventajas más importantes son que:

- es multiplataforma, ya que está escrito en Java,
- utiliza ficheros de construcción escritos en XML (*build.xml*), frente a la sintaxis críptica de *make*,
- puede extenderse fácilmente creando nuevas tareas a medida.

El hecho de ser multiplataforma hace que Ant añada una primera capa de abstracción que aísla al desarrollador respecto a la máquina en que esté trabajando. Un mismo proyecto Ant se comportará igual en el ordenador de un desarrollador que en el entorno de producción: no hay que cambiar las rutas de ficheros, ni depender de un entorno de desarrollo concreto para montar una aplicación, ni hacer quinientas cosas para realizar el proceso de construcción (se realiza todo con un solo clic...).

Esta independencia se ve reforzada por el hecho de que el fichero de configuración está escrito en XML, lo que facilita muchísimo su legibilidad y por tanto el mantenimiento y la extensibilidad de las tareas del proyecto.

9.5.2.2 El fichero de construcción en Ant

Ant busca por defecto el fichero *build.xml* en el directorio desde donde se ha invocado (puede cambiarse con la opción *–buildfile*). Este archivo contiene toda la información referente a las acciones que se realizan para la construcción de un proyecto: compilar, crear directorios, eliminarlos, hacer despliegues de aplicaciones, etc. Dichas acciones en el ámbito de Ant se conocen como tareas. Las tareas se agrupan en el fichero *build.xml* en bloques funcionales, llamados *targets*, que son los que se le pide a Ant que ejecute.

El fichero *build.xml* también indica las dependencias de unas acciones con otras, así como los pasos a seguir cuando se ejecuta una tarea. Cada fichero de construcción contiene un único proyecto, que se descompone en uno o más *targets* y éstas en cero o más tareas. Un ejemplo de dicho fichero puede ser éste:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<project basedir="." default="all" name="test">
	<property environment="env" />

	<!-- borra el directorio build -->
	<target name="clean">
		<delete dir="build"/>
	</target>

	<!-- crea los directorios build/classes y build/lib -->
	<target name="prepare">
		<mkdir dir="build/classes" />
		<mkdir dir="build/lib" />
	</target>

	<!-- define un grupo de archivos que pueden ser usados en el resto de tareas -->
	<path id="jars">
		<fileset dir="${env.CATALINA_HOME}/common/lib" includes="servlet.jar" />
	</path>

	<!-- compila las clases de la carpeta src y las deja en el directorio
build/classes;
	posteriormente monta un jar con todos los archivos .class que hay en
build/classes y
	lo deja en build/lib -->
	<target name="javac">
		<javac classpathref="jars" destdir="build/classes" srcdir="src"/>
		<jar destfile="build/lib/microf.jar" basedir="build/classes"
		includes="microf/**/*.class" />
	</target>

	<!-- ejecuta el resto de targets en el orden indicado -->
	<target depends="clean, prepare, javac" name="all">
	</target>
</project>
```

9.5.2.3 Extendiendo Ant

Para ello se creará una clase que extienda la clase *org.apache.tools.ant.Task*. La ejecución de la tarea se implementa mediante un método *execute* que lanza una excepción *BuildException* en caso de error.

Por cada atributo de la tarea se escribe un método *setter*. El tipo de atributo puede ser un tipo primitivo, *String*, *File*, *Class* o cualquier otro tipo que contenga un constructor con un único parámetro de tipo *String*. A continuación se muestra un ejemplo muy sencillo de una tarea Ant creada a medida:

```
import org.apache.tools.ant.Task;
public class MyEcho extends Task {
        private String message;
        public void setMessage(String m) {
                this.message= m;
        }
        public void execute() throws BuildException {
                log(message);
        }
}
```

Por último se añade al sistema la nueva tarea, que puede hacerse o bien añadiendo un elemento *<taskdef>* al proyecto, o bien de manera permanente, añadiendo la tarea al fichero *defaults.properties* en el paquete *org.apache.tools.ant.taskdefs*.

9.5.3 Maven

9.5.3.1 Qué no es Maven

Características principales de Maven:

- Maven es una herramienta de *sites* y documentación.
- Maven extiende Ant para permitirte bajar dependencias.
- Maven son una serie de *scripts* reusables para gestionar dependencias.

Aunque Maven hace todas estas cosas, no son las únicas características que tiene y sus objetivos son bastante diferentes.

Maven promueve las *best practices* pero algunos proyectos no cuadran bien con estos ideales por razones históricas. Mientras que Maven está diseñado para ser flexible, hasta cierto punto, en estas situaciones y a las necesidades de distintos proyectos, no puede recoger cada situación sin comprometer la integridad de sus objetivos.

Si te decides a usar Maven y tienes una estructura de construcción inusual que no puedes reorganizar, tal vez tengas que olvidarte de algunas características de Maven o incluso del uso del mismo Maven.

9.5.3.2 De Ant a Maven

Ant añade sobre otras plataformas de construcción el hecho de ser multiplataforma y que su sintaxis está escrita en XML, siendo muy sencilla de entender (al contrario que por ejemplo en *make*). Permite hacer construcciones multi-plataforma de un modo estándar y elegante. Pero no todo es perfecto:

- Hay que encajar librerías externas al programa (dependencias), saber poner los distintos descriptores, *properties*, *jars*, etc. en el directorio adecuado... Esto se traduce en que el fichero *build.xml* empieza a crecer de manera desproporcionada, con tareas que son un refrito de otras y que luego se modifican.
- También está el problema del *classpath*: diferentes versiones de diferentes librerías, en diferentes sitios, en diferentes estaciones de trabajo. Y después se oyen cosas del estilo de "¡pero si en mi ordenador compila!"

Maven adopta un punto de vista distinto, ya que fuerza a trabajar de tal modo que se desarrollen módulos (conocidos en el ámbito de Maven como *artifacts*: *WARs*, *JARs*, etc.) que dependan de versiones concretas de librerías, que a su vez se encuentran disponibles en un repositorio común; del mismo modo, estos *artifacts* se publican en el repositorio, de tal manera que estén disponibles para el resto de desarrolladores. Se acabó el problema del *classpath* y se acabó el problema del uso de versiones distintas de la misma librería.

El primer concepto básico de desarrollo en Maven es el proyecto. Un proyecto es cualquier directorio que contenga un archivo *project.xml*. Este archivo se conoce con el nombre de POM (*Project Object Model*), y contiene toda la información y estructura acerca del proyecto (nombre del proyecto, tipo, versión, autor, dependencias, etc.). Un fichero *project.xml* tiene una pinta parecida a esta:

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
        <pomVersion>3</pomVersion>
        <id>maven_dummy_project</id>
        <artifactId>maven_dummy_project_demo</artifactId>
        <name>maven dummy project</name>
        <groupId>metaware</groupId>

        <currentVersion>0.0.1</currentVersion>
        <organization>
                <name>Metaware Inc</name>
                <url>http://metaware-inc.wiki.mailxmail.com/</url>
                <logo>http://maven.apache.org/images/jakarta-logo-blue.gif</logo>
        </organization>

        <inceptionYear>2002</inceptionYear>
        <package>com.metaware.inc</package>
        <logo>http://maven.apache.org/images/maven.jpg</logo>

        <description>A collection of example projects showing how to use maven in
        different situations</description>
        <shortDescription>How to use maven in different situations</shortDescription>
        <url>http://maven.apache.org/reference/plugins/examples/</url>
        <issueTrackingUrl>http://nagoya.apache.org/scarab/servlet/scarab/</issueTrackingUrl>

        <siteAddress>jakarta.apache.org</siteAddress>
        <siteDirectory>/www/maven.apache.org/reference/plugins/examples/</siteDirectory>
        <distributionDirectory>/www/maven.apache.org/builds/</distributionDirectory>

        <repository>
```

```
            <connection>scm:cvs:pserver:anoncvs@cvs.apache.org:/home/cvspublic:
            mavenplugins/ examples</connection>
            <url>http://cvs.apache.org/viewcvs/maven-plugins/examples/</url>
        </repository>
        <mailingLists/>
        <developers/>
        <dependencies>
            <dependency>
                <groupId>commons-beanutils</groupId>
                <artifactId>commons-beanutils</artifactId>
                <version>1.6.1</version>
                <type>jar</type>
                <properties>
                    <war.bundle>true</war.bundle>
                </properties>
            </dependency>
        </dependencies>

        <build>
            <nagEmailAddress>turbine-maven-dev@jakarta.apache.org</nagEmailAddress>
            <sourceDirectory>src/java</sourceDirectory>

            <unitTestSourceDirectory>src/test</unitTestSourceDirectory>
            <unitTest>
                <includes>
                    <include>**/*Test.java</include>
                </includes>
                <excludes>
                    <exclude>**/NaughtyTest.java</exclude>
                </excludes>
            </unitTest>

            <resources>
                <resource>
                    <directory>src/conf</directory>
                    <includes>
                        <include>*.properties</include>
                    </includes>
                    <filtering>false</filtering>
                </resource>
            </resources>

        </build>
</project>
```

Maven debe ser visto como un *framework*: es un entorno de trabajo para la gestión de proyectos y despliegue de aplicaciones. Consta de un núcleo que ejecuta distintas acciones o *goals*, el equivalente de los *targets* en Ant. Dichos *goals* se encuentran distribuidos en *plugins*. Por ejemplo, cuando ejecutamos *maven jar:compile*, estamos ejecutando el *goal compile* del *plugin jar*.

La versatilidad de Maven se encuentra en la cantidad ingente de *plugins* que tiene disponible y que permiten hacer casi cualquier cosa. Además de los *plugins* con los que Maven viene "de serie", encontramos gran cantidad de *plugins* de terceros y, si con eso no basta, siempre se puede desarrollar uno que realice las acciones que se necesiten en un momento dado.

El comportamiento de los *plugins* de Maven es personalizable: hay un fichero (opcional dentro de un proyecto Maven) que se usa para conseguir este objetivo, se trata del archivo *maven.xml*. En él se pueden definir los *goals* y añadirles *pre-goals* y *post-goals*. Un ejemplo de dicho fichero puede ser éste:

```
<project default="foobar-dist" xmlns:m="jelly:maven">
        <preGoal name="java:compile">
            <attainGoal name="xdoclet:webdoclet"/>
        </preGoal>
```

```
        <goal name="foobar-dist">
                <attainGoal name="war:install" />
        </goal>

        <postGoal name="war:war">
                <mkdir dir="${test.result.dir}"/>
                <echo>Creating directories</echo>
        </postGoal>
</project>
```

El último concepto clave es el de repositorio, que es dónde se guardan los *artifacts* de los cuales depende el proyecto. Este repositorio puede ser local o remoto. La idea es que a través del repositorio los desarrolladores compartan todas las librerías. Cada proyecto genera sus *artifacts* y los publica (instalar en Maven) en el repositorio remoto, al que accede el resto del equipo de desarrollo para recogerlos y poder usarlos de un modo estandarizado.

La estructura funcional de Maven puede representarse gráficamente de este modo:

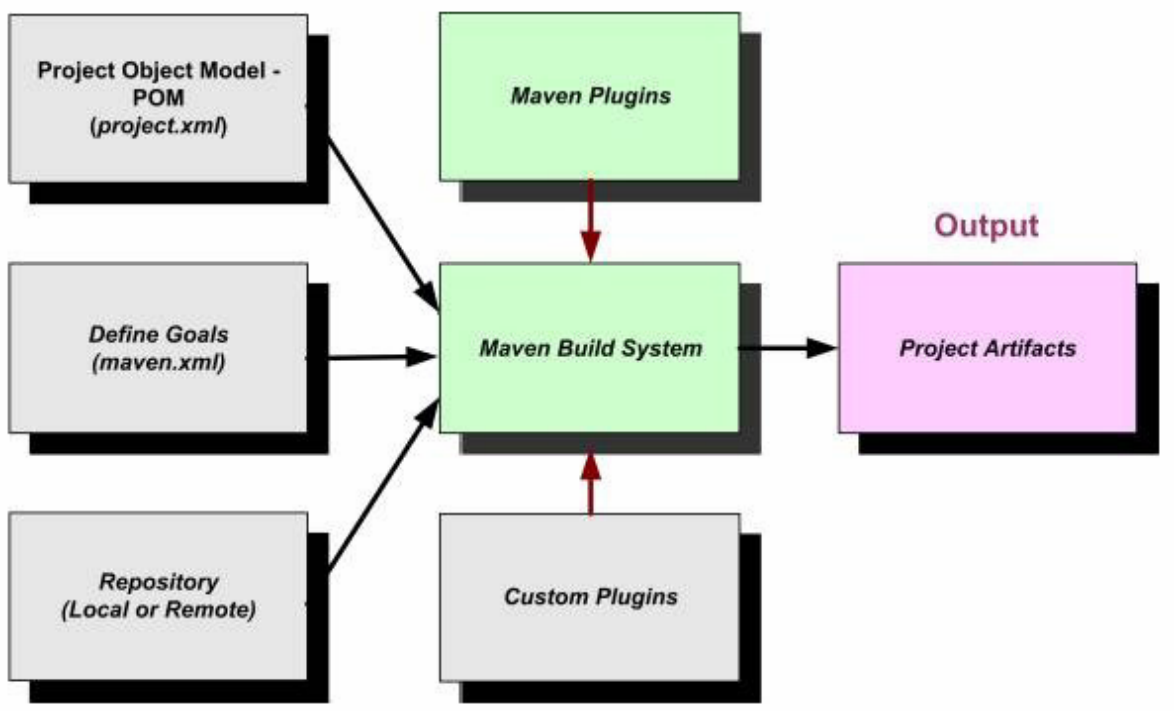


Ilustración 70: Estructura funcional de Maven

Los cuadros grises son dados por el desarrollador (son obligatorios el POM y el repositorio, los otros dos son opcionales), los verdes por Maven y el rosa es el resultado final del proyecto Maven, los *artifacts* generados, uno por proyecto Maven.

Maven por tanto está orientado al trabajo en equipo, consigue que:

- el problema de las distintas dependencias entre proyectos desaparezca
- un mismo proyecto funcione en distintos entornos de desarrollo y/o sistemas operativos sin tocar el código (como mucho ajustando propiedades en un fichero de configuración, y aún así es raro tener que tocar este fichero más de una vez para configurar todos los proyectos)
- el trabajo de un desarrollador se integre de modo transparente con el del resto

- se maximice la cohesión del código y se minimice su acoplamiento, debido al uso intensivo del repositorio Maven
- se facilite la reutilización del código

9.5.3.3 Maven y las metodologías ágiles

Los proyectos en Maven soportan herencia, de tal modo que es posible definir un proyecto "padre", que haga de plantilla del resto de subproyectos definidos dentro de éste. De esta manera, cada subproyecto hereda las propiedades del proyecto raíz. Siguiendo esta filosofía, en Maven está el concepto de Multi-Project y la herramienta *Maven Reactor*, que se encarga de, dado un conjunto de *project.xml*, calcular las dependencias entre ellos y de ejecutarlos en el orden correcto. Además también es posible parametrizar el *Reactor* vía el fichero *maven.xml*, con lo que se pueden hacer auténticas virguerías.

Y aún hay más sobre la potencia de Maven en la integración entre distintos proyectos, dado que se integra con herramientas de integración continua como *Cruise-Control* mediante el *plugin* adecuado. La integración continua es una práctica que forma parte de las llamadas metodologías ágiles. Propuesta por Martin Fowler, consiste en hacer integraciones (compilación y ejecución de tests) automáticas de un proyecto lo más a menudo posible, para así poder detectar los fallos cuanto antes.

El hecho de poder trabajar con varios proyectos independientes pero relacionados entre sí favorece enormemente la refactorización del código, así como poder orientar el desarrollo cara a poder cumplir las llamadas "*small releases*". Además, dentro del fichero *project.xml* se pueden definir dónde están las baterías de tests unitarios (ya sean a través de JUnit o de Cactus) que se ejecutarán nada más terminar de compilar el proyecto.

9.5.3.4 Extendiendo Maven 1.0

Maven 1.0.x se extiende a través de nuevos *plugins* que son escritos en Jelly, un lenguaje de *scripting* basado en xml. Con dicho lenguaje escribimos nuevos *goals* en el fichero *maven.xml*. Desde este fichero también se pueden invocar tanto tareas Ant como métodos de clases que hayamos escrito para tal efecto. Ejecutando el *goal Maven plugin:install* tendremos disponible el *plugin* para todos los proyectos. Ejecutando *maven site* se genera toda la documentación asociada al sitio Un ejemplo de un pequeño *plugin* Maven puede ser éste:

```
<project
        xmlns:j="jelly:core"
        xmlns:m="jelly:maven"
        xmlns:ant="jelly:ant"
        xmlns:util="jelly:util"
        xmlns:artifact="artifact">

        <goal name="war-distribution:pack-depedencies">
                <!--Remove previous libs-->
                <delete dir="${maven.build.dir}/dependencies/lib" />

                <!--Copy dependencies to bundle-->
                <j:forEach var="lib" items="${pom.artifacts}">
```

```
                        <j:set var="dependency" value="${lib.dependency}"/>
                        <j:if test="${dependency.getProperty('war.bundle') != null}">
                                <ant:copy file="${lib.path}"
                                        toDir="${maven.build.dir}/dependencies/lib/${depend
                                        ency. getProperty('war.bundle')}" />
                        </j:if>
                </j:forEach>

                <!--Build tar-->
                <tar destfile="${maven.build.dir}/${pom.artifactId}-
                ${pom.currentVersion}.tar"
                        basedir="${maven.build.dir}/dependencies" />

                <echo>Created tar holding dependencies</echo>
        </goal>
</project>
```

9.5.4 Comparación de características de Ant y Maven

	Ant	*Maven*
Instalación	Muy fácil	Muy fácil
Tiempo para empezar un proyecto nuevo	5 minutos	1 minuto (mediante el *plugin genipa* en Maven 1.0 o mediante el *plugin archetype* en Maven 2.0)
Tiempo que se tarda en añadir una nueva funcionalidad	10 minutos para añadir un *target* nuevo	2 minutos en usar un *goal* nuevo
Tiempo de aprendizaje de la herramienta	30 minutos	2 horas
Estructura estándar en los proyectos	No	Sí
Soporte para Multi-Projects	Sí, pero es complicado de implementar	Sí, está orientado a ello
Generación de documentación	No, pero hay muchas herramientas disponibles para ello	Sí
Integración con IDEs	Sí, en casi todos, por no decir en todos (o por lo menos en cualquier IDE que se precie)	A través del *plugin Mevenide*, en Eclipse, Netbeans, JBuilder e IntelliJ IDEA

Ilustración 71: Ant vs Maven

Como se comentó al principio del artículo, una comparación de Maven con Ant nunca será del todo acertada, ya que Maven abarca una funcionalidad mucho mayor que la que pretende Ant. Pero dependiendo del caso puede ser mejor utilizar uno u otro.

9.6 MySQL

MySQL es un sistema de administración de bases de datos. Una base de datos es una colección estructurada de tablas que contienen datos. Esta puede ser desde una simple lista de compras a una galería de pinturas o el vasto volumen de información en una red corporativa. Para agregar, acceder a y procesar datos guardados en un computador, usted necesita un administrador como MySQL Server. Dado que los computadores son muy buenos manejando grandes cantidades de información, los administradores de bases de datos juegan un papel central en computación, como aplicaciones independientes o como parte de otras aplicaciones.

MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

MySQL es software de fuente abierta. Fuente abierta significa que es posible para cualquier persona usarlo y modificarlo. Cualquier persona puede bajar el código fuente de MySQL y usarlo sin pagar. Cualquier interesado puede estudiar el código fuente y ajustarlo a sus necesidades. MySQL usa el GPL (GNU General Public License) para definir que puede hacer y que no puede hacer con el software en diferentes situaciones. Si usted no se ajusta al GLP o requiere introducir código MySQL en aplicaciones comerciales, usted puede comprar una versión comercial licenciada.

MySQL es muy utilizado en aplicaciones Web como MediaWiki o Drupal, en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Su popularidad como aplicación Web está muy ligada a PHP, que a menudo aparece en combinación con MySQL. MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones Web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

Otro de las características de MySQL es su gran número de usuarios (cerca de seis millones), esto es debido principalmente a tres motivos: su licencia open source, por lo cual es un producto gratuito, su compatibilidad con casi todos de los lenguajes de programación actuales y por ser un producto multiplataforma, es decir, puede ser montada en cualquier versión de Linux, Windows o Mac.

9.6.1 Características de la versión 5.0.22 de MySQL

- Un amplio subconjunto de ANSI SQL 99, y varias extensiones.
- Soporte a multiplataforma
- Procedimientos almacenados
- Triggers
- Cursors
- Vistas actualizables
- Soporte a VARCHAR
- INFORMATION_SCHEMA
- Modo Strict
- Soporte X/Open XA de transacciones distribuidas; transacción en dos fases como parte de esto, utilizando el motor InnoDB de Oracle
- Motores de almacenamiento independientes (MyISAM para lecturas rápidas, InnoDB para transacciones e integridad referencial)
- Transacciones con los motores de almacenamiento InnoDB, BDB Y Cluster; puntos de recuperación(savepoints) con InnoDB
- Soporte para SSL
- Query caching
- Sub-SELECTs (o SELECTs anidados)
- Replication with one master per slave, many slaves per master, no automatic support for multiple masters per slave.
- indexing y buscando campos de texto completos usando el motor de almacenamiento MyISAM
- Embedded database library
- Soporte completo para Unicode
- Conforme a las reglas ACID usando los motores InnoDB, BDB y Cluster
- Shared-nothing clustering through MySQL Cluster

9.6.2 Características adicionales

- Usa GNU Automake, Autoconf, y Libtool para portabilidad
- Uso de multihilos mediante hilos del kernel.
- Usa tablas en disco b-tree para búsquedas rápidas con compresión de índice
- Tablas hash en memoria temporales
- El código MySQL se prueba con Purify (un detector de memoria perdida comercial) así como con Valgrind, una herramienta GPL
- Completo soporte para operadores y funciones en cláusulas select y where.
- Completo soporte para cláusulas group by y order by, soporte de funciones de agrupación
- Seguridad: ofrece un sistema de contraseñas y privilegios seguro mediante verificación basada en el host y el tráfico de contraseñas está cifrado al conectarse a un servidor.
- Soporta gran cantidad de datos. MySQL Server tiene bases de datos de hasta 50 millones de registros.
- Se permiten hasta 64 índices por tabla (32 antes de MySQL 4.1.2). Cada índice puede consistir desde 1 hasta 16 columnas o partes de columnas. El máximo ancho de límite son 1000 bytes (500 antes de MySQL 4.1.2).

- Los clientes se conectan al servidor MySQL usando sockets TCP/IP en cualquier plataforma. En sistemas Windows se pueden conectar usando named pipes y en sistemas Unix usando ficheros socket Unix.
- En MySQL 5.0, los clientes y servidores Windows se pueden conectar usando memoria compartida.
- MySQL contiene su propio paquete de pruebas de rendimiento proporcionado con el código fuente de la distribución de MySQL

9.7 Qué es XML.

XML es una tecnología en realidad muy sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores.

XML, con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, más avanzada, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. Así pues, el XML juega un papel importantísimo en este mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable y fácil. Además, XML permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

XML no es un lenguaje, sino varios lenguajes, no es una sintaxis, sino varias y no es una manera totalmente nueva de trabajar, sino una manera más refinada que permitirá que todas las anteriores se puedan comunicar entre si sin problemas, ya que los datos cobran sentido.

9.7.1 Historia del XML.

El XML proviene de un lenguaje que inventó IBM allá por los años 70. El lenguaje de IBM se llama GML (General Markup Language) y surgió por la necesidad que tenían en la empresa de almacenar grandes cantidades de información de temas diversos.

Imaginar por un momento la cantidad de documentación que generaría IBM sobre todas las áreas en las que trabajaba e investigaba, y la cantidad de información que habrá generado hasta hoy. Así pues, necesitaban una manera de guardar la información y los expertos de IBM se inventaron GML, un lenguaje con el que poder clasificarlo todo y escribir cualquier documento para que se pueda luego procesar adecuadamente.

Este lenguaje gustó mucho a la gente de ISO, una entidad que se encarga de normalizar cuantas cosas podáis imaginar para los procesos del mundo actual, de modo que allá por el 86 trabajaron para normalizar el lenguaje, creando el SGML, que no era más que el GML pero estándar (Standar en inglés).

SGML es un lenguaje muy trabajado, capaz de adaptarse a un gran abanico de problemas y a partir de él se han creado los siguientes sistemas para almacenar información.

Por el año 89, para el ámbito de la red Internet, un usuario que había conocido el lenguaje de etiquetas (Markup) y los hiperenlaces creo un nuevo lenguaje llamado HTML, que fue utilizado para un nuevo servicio de Internet, la Web. Este lenguaje fue adoptado rápidamente por la comunidad y varias organizaciones comerciales crearon sus propios visores de HTML y riñeron entre ellos para hacer el visor más avanzado,

inventándose etiquetas como su propia voluntad les decía. Desde el 96 hasta hoy una entidad llamada W3C ha tratado de poner orden en el HTML y establecer sus reglas y etiquetas para que sea un estándar. Sin embargo el HTML creció de una manera descontrolada y no cumplió todos los problemas que planteaba la sociedad global de Internet.

El mismo W3C en el 98 empezó y continúa, en el desarrollo de XML (Extended Markup Language). En este lenguaje se ha pensado mucho más y muchas personas con grandes conocimientos en la materia están trabajando todavía en su gestación. Pretendían solucionar las carencias del HTML en lo que se respecta al tratamiento de la información. Problemas del HTML como:

- El contenido se mezcla con los estilos que se le quieren aplicar.
- No permite compartir información con todos los dispositivos, como pueden ser ordenadores o teléfonos móviles.
- La presentación en pantalla depende del visor que se utilice.

Imagínese, una persona que conoce el HTML y lo difícil que puede llegar a ser entender su código, que tuviese que procesarlo para extraer datos que necesite en otras aplicaciones. Sería muy difícil saber dónde está realmente la información que busca, siempre mezclada entre etiquetas <FONT>, <TABLE>, <TD>, etc. Esto es una mala gestión de la información y el XML la soluciona.

9.7.2 Sintaxis del XML.

Dicen que el XML es un 10% del SGML y de verdad lo es, porque en realidad las normas que tiene son muy simples. Se escribe en un documento de texto ASCII, igual que el HTML y en la cabecera del documento se tiene que poner el texto

```
<?xml version="1.0"?>
```

En el resto del documento se deben escribir etiquetas como las de HTML, las etiquetas que nosotros queramos, por eso el lenguaje se llama XML, lenguaje de etiquetas extendido. Las etiquetas se escriben anidas, unas dentro de otras.

```
<ETIQ1>...<ETIQ2>...</ETIQ2>...</ETIQ1>
```

Cualquier etiqueta puede tener atributos. Le podemos poner los atributos que queramos.

```
<ETIQ atributo1="valor1" atributo2="valor2"...>
```

Los comentarios de XML se escriben igual que los de HTML.

```
<!-- Comentario -->
```

Y esto es todo lo que es el lenguaje XML en sí, aunque tenemos que tener en cuenta que el XML tiene muchos otros lenguajes y tecnologías trabajando alrededor de él. Sin embargo, no cabe duda que la sintaxis XML es realmente reducida y sencilla.
Para definir qué etiquetas y atributos debemos utilizar al escribir en XML tenemos que fijarnos en la manera de guardar la información de una forma estructurada y ordenada. Por ejemplo, si deseamos guardar la información relacionada con una película en un documento XML podríamos utilizar un esquema con las siguientes etiquetas.

```
<?xml version="1.0"?> <PELICULA nombre="El Padrino" año=1985> <PERSONAL>
<DIRECTOR nombre="Georgie Lucar"/> <INTERPRETE nombre="Marlon Brando" interpreta-
a="Don Corleone"/> <INTERPRETE nombre="Al Pacino" interpreta-a="Michael
Corleone"/> </PERSONAL> <ARGUMENTO descripción="Pelicula de mafias sicilianas en
Estados Unidos"/> </PELICULA>
```

9.7.3 Contenidos: DTD o XML Schema

Un documento XML puede contener muchos tipos de información. Es decir, pueden haber muchos lenguajes escritos en XML para cualquier colectivo de usuarios.

Como vemos, se pueden crear infinitos lenguajes a partir del XML. Para especificar cada uno de los usos de XML, o lo que es lo mismo, para especificar cada uno de los sublenguajes que podemos crear a partir de XML, se utilizan unos lenguajes propios.

Son unos lenguajes que sirven para definir otros lenguajes, es decir, son metalenguajes. Los definen especificando qué etiquetas podemos o debemos encontrarnos en los documentos HTML, en qué orden, dentro de qué otras, además de especificar los atributos que pueden o deben tener cada una de las etiquetas.

Hay dos metalenguajes con los que definir los lenguajes que podemos obtener a partir de XML, el DTD y el XML Schema.

El DTD, Definition Type Document, tiene una sintaxis especial, distinta de la de XML, que es sencilla, aunque un poco rara si nunca hemos visto un documento similar.

Para evitar el DTD, que tiene una sintaxis muy especial, se intentó encontrar una manera de escribir en XML la definición de otro lenguaje XML. Se definió entonces el lenguaje XML Schema y funciona bien, aunque puede llegar a ser un poco más complicado que especificarlo en DTD. Simplemente nos ahorramos de aprender un nuevo lenguaje con su sintaxis particular.

9.7.4 Diseño: CSS o XSL.

Para cada documento XML que se desee presentar en pantalla formateado de la manera que deseemos se tiene que escribir una hoja de estilos o similar.

También tenemos dos posibles lenguajes con los que formatear los textos de un documento XML para poder verlo por pantalla. La primera posibilidad es el CSS y la segunda, el XSL, bastante más avanzada.

9.7.5 Programación: SAX o DOM.

Si queremos realizar acciones con nuestros datos escritos en XML tenemos también mucho camino ya implementado. El W3C ha especificado dos mecanismos para acceder a documentos XML y trabajar con ellos. Se tratan simplemente de unas normas que indican a los desarrolladores la manera de acceder a los documentos. Estas normas incluyen una jerarquía de objetos que tienen unos métodos y atributos con los que tendremos que trabajar y que nos simplificarán las tareas relativas al recorrido y acceso a las partes del documento.

Estos dos mecanismos se denominan **SAX y DOM**. SAX se utiliza para hacer un recorrido secuencial de los elementos del documento XLM y DOM implica la creación

de un árbol en memoria que contiene el documento XML, y con él en memoria podemos hacer cualquier tipo de recorrido y acciones con los elementos que queramos.

9.7.5.1 El API SAX

SAX define un API para un analizador de archivos *XML* basado en eventos. Estar "basado en eventos" significa que el analizador lee un documento *XML* desde el principio hasta el final, y cada vez que reconoce una sintaxis de construcción, se lo notifica a la aplicación que lo está ejecutando. SAX notifica a la aplicación llamando a los métodos del interface `ContentHandler`. Por ejemplo, cuando el analizador encuentra un símbolo ("<"), llama al método `startElement`; cuando encuentra caracteres de datos, llama al método `characters`; y cuando encuentra un símbolo ("</"), llama al método `endElement`, etc.

Cuando comenzamos a pensar en nuestro proyecto, se nos planteo la duda de que API utilizar para la lectura de *XML's*, teníamos dos opciones: SAX o DOM. DOM es un conjunto de interfaces para construir una representación de objeto, en forma de árbol, de un documento XML analizado, al ser más complicado que SAX, y debido a que los documentos de configuración que íbamos a crear eran sencillos y no muy extensos, nos decidimos por este último.

9.7.5.2 El API JDOM

JDOM es un API para leer, crear y manipular documentos XML de una manera sencilla y muy intuitiva para cualquier programador en Java, en contra de otras APIs tales como DOM y SAX, las cuales se idearon sin pensar en ningún lenguaje en concreto, de ahí que resulte un poco incomoda su utilización. Precisamente esta comodidad que ofrece JDOM frente a SAX o DOM, es la que nos ha llevado a utilizarlo en nuestro proyecto.

El *xml* se forma con objetos *Element*, que representan a los elementos de un *xml* (*cada tag* que se abre y se cierra es un elemento). Para añadir un elemento dentro de otro elemento se utiliza el método *addContent (Element element)*. Para añadir un atributo a un elemento se utiliza el método *setAttribute (Attribute atributte)*, de donde podemos deducir que la clase *Attribute* representa a un atributo de un elemento del *xml*. Por último tenemos el método *setText (String text)*, que está en la clase *Element* y sirve para añadir un texto al elemento. Estos son los métodos y clases básicas, con ellos vamos creando una especie de árbol de elementos que representa el documento, y que se convierte fácilmente en un Archivo *xml.*

10 Librerías utilizadas

A continuación mostramos una tabla con las librerías utilizadas para la implementación de la aplicación, con una breve descripción de la misma y la utilización dentro de nuestro proyecto.

Nombre	*Versión*	*Descripción*	*Módulo de Uso*
ant	1.5.3-1	Jar que recoge la funcionalidad básica de Ant	Todos
antlr	2.7.6	ANTLR: ANother Tool for Language Recognition (Otra herramienta para el Reconocimiento del Idioma). Proporciona un framework para construir reconocedores, compiladores, y traductores de las descripciones gramaticales que contienen Java, C + +, o C#.	Necesaria para las librerías groovy-all e hibernate
asm	1.5.3	Biblioteca para la persistencia de Hibernate	Necesaria para hibernate
asm-attrs	1.5.3	Biblioteca para la persistencia de Hibernate	Necesaria para hibernate
bsf	2.4.0	BSF: Bean Scripting Framework es un conjunto de clases java que permite escribir JSPs en otros lenguajes distintos de java proporcionando acceso a la librería de clases java.	Necesaria para corregir los exámenes
cglib-nodep	2.1.3	CGLIB es una poderosa librería de generación de código de alto rendimiento y calidad. Es usada para extender clases Java e implementa interfaces en tiempo de ejecución.	Necesaria para hibernate
commons-beanutils	1.7.0	Librería de Apache que facilita el uso de beans de manera dinámica (sin compilado).	Todos
commons-codec	1.3	Librería de Apache	Todos
commons-collections	3.1	Librería de Apache que extiende o aumenta la librería de Colecciones de Java.	Todos
commons-dbcp	1.2.1	Librería de Apache que aporta servicios de conexión a la base de datos.	Módulo de persistencia
commons-digester	1.6	Librería de Apache que se utiliza para mapear XML a objetos Java	Módulo de la interfaz gráfica
commons-discovery	0.2	Librería de Apache que proporciona facilidades para instanciar clases en general, y para la gestión del ciclo de vida da clases singleton (factoría)	Necesario para el patrón factoría
commons-el	1.0	Librería de Apache	Todos
commons-id	0.1	Librería de Apache	Todos
commons-lang	2.1	Librería de Apache	Todos
commons-logging	1.0.4	Librería de Apache para la generación de logs	Todos
commons-logging-api	1.0.3	Librería de Apache para la generación de logs	Todos

Nombre	*Versión*	*Descripción*	*Módulo de Uso*
commons-pool	1.2	Librería de Apache que proporciona una reunión de objetos.	Todos
dom4j	1.6.1	Librería que ofrece soporte para API's de procesamiento de XML, en nuestro caso para SAX.	Módulo de la interfaz gráfica
ehcache	1.2.3	Ehcache es generalmente para java cache distribuidas para caching de propósito general, J2EE y contenedores ligeros.	Modulo de la persistencia
el-api	1.0		
el-ri	1.0	Librería que realiza la conexión con Tomcat	
facestrace	0.8.1	Librería de SVN.	Repositorio
hibernate	3.2.1	Librería de Hibernate	Módulo de la persistencia
hsqldb	1.8.0.7	Librería de HipersonicSQLDB	Módulo de la persistencia
javolution-jdk1.4	4.0.2	Librería para combinar el uso de java con XML	Todos
joda-time	1.4	Librería que proporciona un reemplazo (suplente) de calidad para la fecha Java y las clases time.	Todos
jsf-facelets	1.1.11	Librería de distribución de JSF en Open Source	Módulo de la interfaz gráfica
jstl	1.1.2	Proyecto de Apache que contiene un repositorio de librerías de etiqueta JSP de encargo y proyectos asociados.	Módulo de la interfaz gráfica
jta	1.0.1B	Librería para la Java Transaction API	Módulo de la persistencia
log4j	1.2.9	Proyecto de apache para la generación de logs	Todos
myfaces-api	1.1.4	Librería de JSF	Módulo de la interfaz gráfica
myfaces-impl	1.1.4	Librería de JSF	Módulo de la interfaz gráfica
spring	2.0	Librería de Spring.	Módulo de la persistencia
standard	1.1.2	Librería de Apache de Tag Libraries	Módulo de la interfaz gráfica
xercesImpl	2.0.2		
xml-apis	1.0.b2	Contiene las clases java que necesita JDOM para dar formato al XML	

Ilustración 72: Librerías utilizadas

11 Integrantes del Proyecto y metodología de trabajo

11.1 Participantes

El grupo se compone de cinco miembros, tres alumnos que han desarrollado el proyecto:

- Natalia Rivera Tolosa
- Alberto Sánchez San Felipe
- David Duce pastor

Más dos profesores-directores, que han guiado a los alumnos:

- Baltasar Fernández Manjón
- Iván Martínez Ortiz

11.2 Jefe de equipo

Al tratarse de un grupo de tres personas, no ha habido división jerárquica entre los miembros del grupo, aunque si que ha habido un jefe de grupo encargado de ponerse en contacto con los profesores-directores (Alberto Sánchez San Felipe).

11.3 Equipo de desarrollo

Descentralizado democrático
-no tiene un jefe permanente
-se nombre un jefe en función de cada tarea
-las decisiones, problemas y enfoques se llevan a consenso del grupo
-la comunicación entre los miembros del equipo es horizontal

11.3.1 Seguimiento y reuniones

El seguimiento y las reuniones entre los tres alumnos y los directores profesores se han llevado a cabo los viernes a las 15.00, en el despacho 414 de la cuarta planta de la facultad de informática correspondiente al profesor Baltasar Fernández Manjón.

Cuando ha sido necesario también se han realizado reuniones virtuales a través del Messenger, Gmail o mediante Skype.

11.3.2 Comunicación entre el grupo

La comunicación entre los tres alumnos del grupo y los profesores-directores se ha llevado a cabo mediante correos electrónicos, mediante Messenger y mediante Skype.

11.4 Gestión de archivos

Mediante SVN Repository

12 Requisitos de la aplicación

12.1 Hardware necesario

Para el desarrollo de la aplicación se requiere como configuración mínima un Pentium III a 800 Mhz.

12.2 Software necesario

El software requerido para el desarrollo del proyecto es:

- Eclipse 3.2.1
- Tomcat 5.5.17
- JDK 1.5.0_09
- SVN
- Plugin tomcatPluginV31
- Plugin subclipse_1.0.1
- Ant 1.6.5
- Maven 2
- Librerías mencionadas anteriormente
- Mozilla Firefox
- Microsoft Office
- MySQL Server 5.0

13 Conclusiones

El proyecto eQTI de Sistemas Informáticos nos ha servido para introducirnos en un proyecto de la vida real, un proyecto en el que se ha seguido un estándar para el desarrollo de la aplicación y en el que hemos empleado y programado las tecnologías más punteras de la actualidad, en cuanto a lo que el Software se refiere (JSF, Spring, Hibernate,…), todas ellas de Software Libre, como cada día más empresas implantan en sus proyectos.

El uso de estas tecnologías, desconocidas hasta el momento por nosotros, supuso en un principio un gran inconveniente, ya que nos obligó a realizar una búsqueda exhaustiva de información a través del medio que más nos ha ayudado en el desarrollo del trabajo, Internet. La decisión de emplear este medio de información como mayor fuente de conocimiento, se tomó debido a la falta de documentación escrita y nuestra facilidad de acceso a la red.

Aunque el período de búsqueda de información más amplio en el tiempo fue antes del comienzo de la generación de código, esta práctica no ha cesado a lo largo de todo el desarrollo del proyecto, ya que nos ha sido imprescindible a medida que avanzábamos, sobretodo para la resolución de los problemas surgidos durante la implementación.

Debido a la dimensión del proyecto, contamos con el apoyo y la ayuda de Iván Martínez, quien, durante todo el proceso de desarrollo, nos ha facilitado su ayuda y conocimientos técnicos para poder resolver algunos problemas derivados de la complejidad de la aplicación. Este apoyo, nos ha facilitado algunas tareas y hemos podido comprobar la importancia de tener alguien cercano con grandes conocimientos en la materia tratada, ya que algunas de nuestras dudas podrían ser de fácil solución con los conocimientos adecuados, pero difíciles de encontrar mediante búsquedas en Internet o documentación escrita.

Además, a pesar del gran esfuerzo que ha llevado, el haber realizado un proyecto de estas características nos ha ayudado a dar el paso de maduración que necesitábamos para afrontar, el día de mañana, la vida laboral fuera de la Universidad.

13.1 Estado actual del proyecto

El proyecto eQTI se ha desarrollado según el estándar de IMS QTI versión 2.1 y cuyo estado actual es el siguiente:

- Ejecución y evaluación de los tipos de preguntas: Verdadero/Falso, Elección Simple, Elección Múltiple, Asociación, Entrada de texto y Elección en Línea.

- Creación de los tipos de preguntas: Verdadero/Falso, Elección Simple, Elección Múltiple, Asociación, Entrada de texto y Elección en Línea.

- Edición de los tipos de preguntas: Verdadero/Falso, Elección Simple, Elección Múltiple, Asociación, Entrada de texto y Elección en Línea.

- Creación y edición de test.
- Ejecución y evaluación de test.
- Máquina de estados del ítem.
- Máquina de estados del test.
- Distintas funcionalidades según el tipo de usuario: alumno (ejecución y evaluación de los recursos creados) y tutor (creación y edición de recursos).

.

13.2 Futuras líneas de desarrollo

Estas son algunas de las posibles ampliaciones y mejoras que se nos ocurren para mejorar la aplicación:

- Hacer exportables los test realizados en la parte del tutor, pudiéndolos pasar a formato pdf o pudiendo imprimir varias versiones de los test creados con diferente organización tanto de las preguntas como de las respuesta de cada una de las preguntas.
- Ampliar el tipo de preguntas que se pueden realizar con la aplicación pudiendo añadir tipos de preguntas como: respuestas basadas en elementos multimedia, respuestas ordenadas o basadas en el desplazamiento de barras o imágenes.
- Añadir la funcionalidad de poder importar preguntas y test creadas en formato XML por otras aplicaciones que sigan la versión implementada del estándar QTI.
- Añadir la navegación a través de las preguntas de un test durante el examen y permitir la posibilidad de dejar preguntas sin contestar en el mismo.
- Añadir ítems adaptativos.
- Permitir que el tutor cuando este creando o editando una pregunta pueda añadir elementos multimedia.
- Permitir la administración de nuevos “Sites”.
- Permitir que nuevos alumnos se puedan registrar.
- Permitir que nuevos tutores se puedan registrar.

14 Bibliografía

[1] IMS Question and Test Interoperability Information Model. Version 2.0 Final Specification http://www.imsproject.org/question/qti_v2p0/imsqti_infov2p0.html

[2] IMS Question & Test Interoperability: ASI Best Practice & Implementation Guide - http://www.imsproject.org/question/qtiv1p2/imsqti_asi_bestv1p2.html

[3] IMS Question & Test Interoperability: ASI XML Binding Specification - http://www.imsproject.org/question/qtiv1p2/imsqti_asi_bindv1p2.html

[4] IMS Question and Test Interoperability Overview - http://www.imsproject.org/question/qti_v2p0/imsqti_oviewv2p0.html

[5] IMS Question and Test Interoperability XML Binding - http://www.imsglobal.org/question/qti_v2p0/imsqti_bindv2p0.html#binding_prompt

[6] IMS Question & Test Interoperability Specification http://www.imsproject.org/question/

[7] XDoclet Attribute-Oriented Programming - Tag Reference http://xdoclet.sourceforge.net/xdoclet/tags/hibernate-tags.html

[8] Hibernate and XDoclet - How generate Hibernate mapping files with XDoclet - http://www.downside.ch/hibernate/hibernatecheatsheet-1.4.pdf

[9] Bauer Gaving King C. *Hibernate in Action* - Manning Publications Co. 2005

[10] Hibernate website - http://www.hibernate.org/

[11] Hibernate Reference Documentation version3.1 - http://www.hibernate.org/hib_docs/v3/reference/en/html/

[12] Hibernate - Relational Persistence for Idiomatic Java - http://www.hibernate.org/hib_docs/reference/en/html/

[13] Hibernate Basics - http://www.developer.com/open/article.php/10930_3559931_1

[14] Persistencia de Objetos Java: El Camino hacia Hibernate - http://www.programacion.com/tutorial/hibernate/

[15] Spring Framework - http://www.springframework.org

[16] Introduction to the Spring Framework- http://www.theserverside.com/articles/article.tss?l=SpringFramework

[17] The Spring series - http://www-128.ibm.com/developerworks/views/web/libraryview.jsp?search_by=The+Spring+Series

[18] Object-relation mapping without the container - http://www-128.ibm.com/developerworks/library/j-hibern/?ca=dnt-515

[19] Spring Framework - http://sourceforge.net/projects/springframework

[20] Spring Framework - http://en.wikipedia.org/wiki/Spring_framework

[21] The spring - http://javaboutique.internet.com/tutorials/spring_frame/

[22] Bruce Eckel. *Piensa en Java. Segunda edición.* Pearson Education, S. A. Madrid. 2002

[23] The J2EE 1.4 Tutorial http://java.sun.com/j2ee/1.4/docs/tutorial/doc/

[24] JavaServer Faces HTML Tags - http://www.exadel.com/tutorial/jsf/jsftags-guide.html

[25] Html code tutorial - http://www.htmlcodetutorial.com/forms/

[26] Java Server Faces. Constant Field Values - http://java.sun.com/javaee/javaserverfaces/1.1_01/docs/api/constant-values.html

[27] XSL Transformations (XSLT) - Http://www.w3.org/TR/xslt#local-variables

[28] JSF communication - http://balusc.xs4all.nl/srv/dev-j2p-com.html

[29] The Java EE Tutorial - http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

[30] Core JavaServer Faces by David Geary and Cay Horstmann, Sun Microsystems Press 2004. - http://www.horstmann.com/corejsf/

[31] The Apache MyFaces Project - http://myfaces.apache.org/

[32] JSF Core Tag Library - http://myfaces.apache.org/impl/tlddoc/f/tld-frame.html
Sun Developer Network. Java Technology Forums - http://forum.java.sun.com/index.jspa

[33] The J2EE 1.4 Tutorial - http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html (Tema 17)

[34] JSF for nonbelievers: JSF conversion and validation - http://www-128.ibm.com/developerworks/java/library/j-jsf3/index.html

[35] Yang Shen, Derek , (1999) *Integración de JSF, Spring e Hibernate para crear una Aplicación Web del Mundo Real* - http://www.programacion.com/tutorial/jap_jsfwork/3/

[36] Apache Ant Project - http://ant.apache.org/

[37] Apache Ant - http://es.wikipedia.org/wiki/Ant

[38] Introducción a Ant - http://www.javahispano.org/articles.article.action?id=31

[39] Apache Maven Project - http://maven.apache.org/

[40] Maven - http://es.wikipedia.org/wiki/Maven

[41] Apache Maven Simplifica el Proceso de Construcción - Incluso más que Ant - http://www.programacion.com/java/tutorial/jap_maven/

[42] Ant & Maven http://metaware-inc.wiki.mailxmail.com/AntMaven

[43] HSQL database engine - http://www.hsqldb.org/

[44] HSQLDB User Guide - http://hsqldb.sourceforge.net/web/hsqlDocsFrame.html

[45] Eclipse and HSQLDB - http://www-128.ibm.com/developerworks/opensource/library/os-echsql/

[46] G. Booch, J. Rumbaugh, I. Jacobson. *El lenguaje unificado de modelado*. Addison Wesley iberoamericana. Madrid. 1999

[47] A. Gutiérrez, R. Martínez. *XML a través de ejemplos*. Ra-Ma. 2001.

15 Manual de usuario

15.1 Manual de usuario del alumno

15.1.1 Pantalla de inicio de sesión

Para que un alumno utilice la aplicación debe estar dado de alta y registrado en el sistema con role de alumno, una vez que esta registrado el alumno debe de ir a la pantalla inicial del sistema donde debe introducir su usuario y su contraseña y a continuación presionar el botón de login.

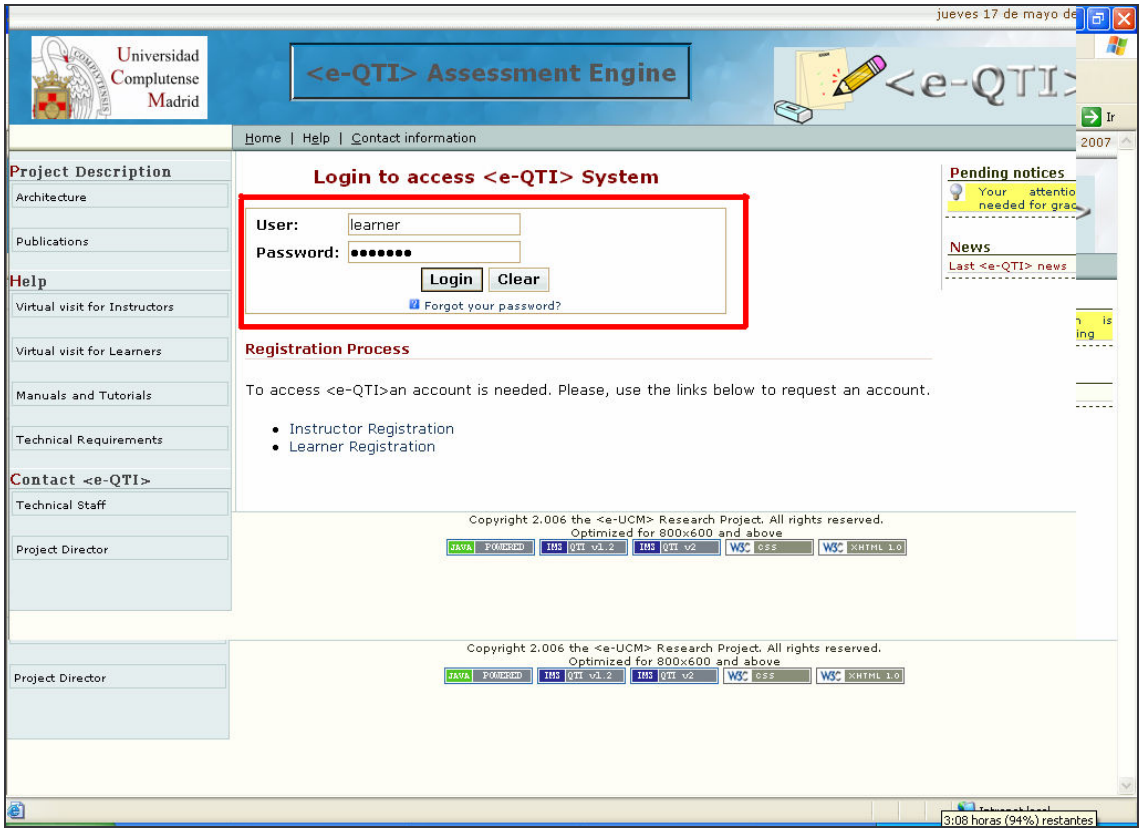


Ilustración 73: Pantalla de inicio de sesión

15.1.2 Pantalla de elección de Site

La siguiente pantalla que se le muestra al alumno, es la pantalla donde se visualizan los "sites" o espacios habilitados por los diferentes tutores para realizar los test y preguntas que tiene que realizar el alumno. En este punto el alumno debe elegir uno de los espacios habilitados presionando el enlace correspondiente del "site".

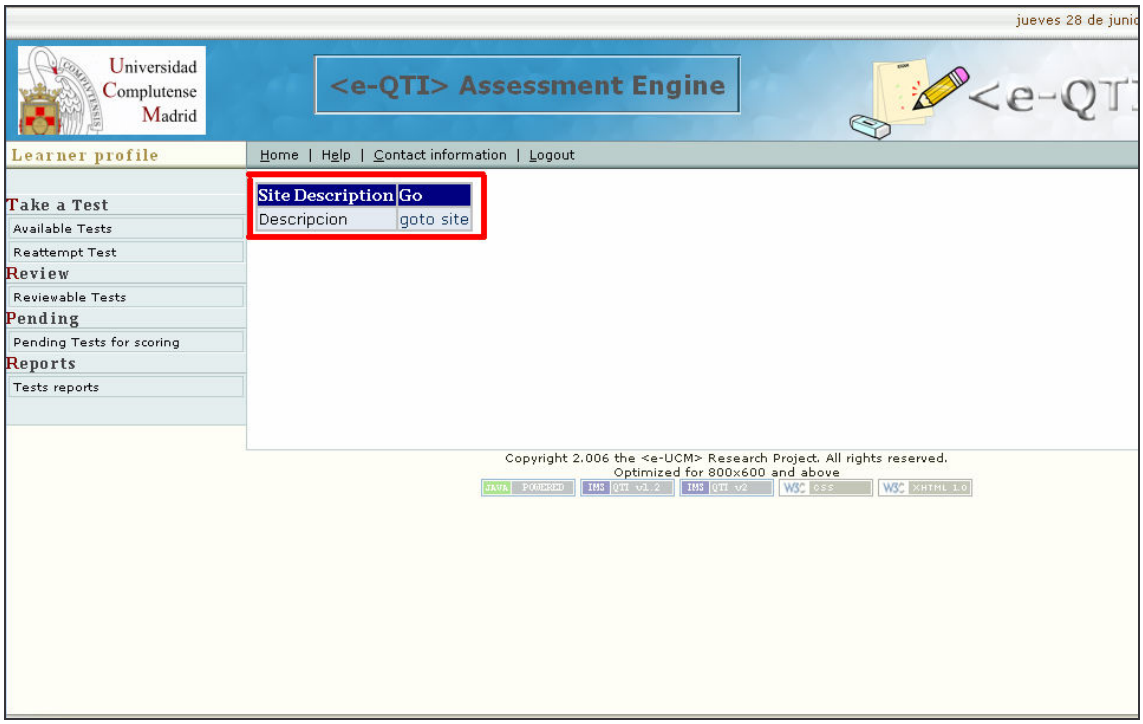


Ilustración 74: Pantalla de elección de Site

15.1.3 Pantalla de elección de tests y preguntas

En el siguiente paso se muestra el contenido del "site" habilitado para el alumno, en el se muestran todos los tests y preguntas habilitadas. Para que un alumno comience la ejecución de un test o pregunta debe presionar el enlace "start test" o "start item" del test o pregunta elegido.

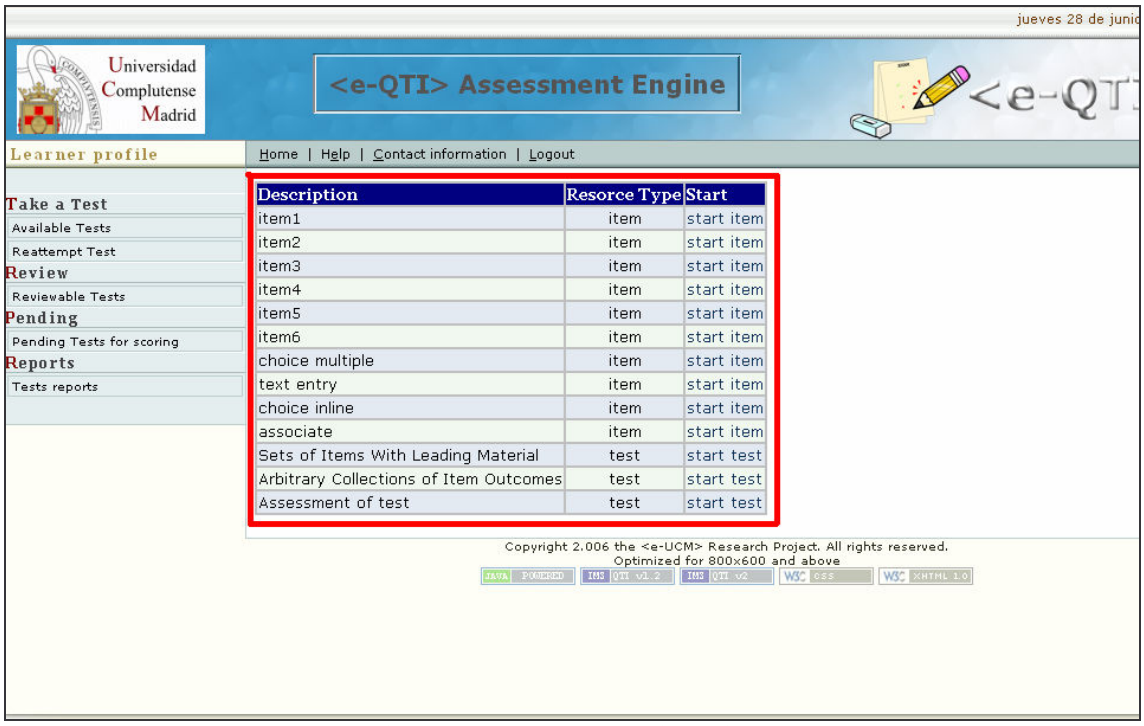


Ilustración 75: Pantalla de elección de ítems y tests

15.1.4 Pantalla resumen de corrección

15.1.4.1 Corrección de test

Si el alumno ha elegido un test, inmediatamente se le va a mostrar la primera pregunta del test y según vaya contestando a la pregunta mostrada en pantalla se le irán mostrando las preguntas restantes del test. Por último, cuando se haya respondido a la última pregunta del test se le mostrará al alumno una pantalla resumen con los resultados obtenidos en cada una de las preguntas y la puntuación total obtenida en el test. Para salir de la pantalla resumen se debe presionar el enlace "End Review".

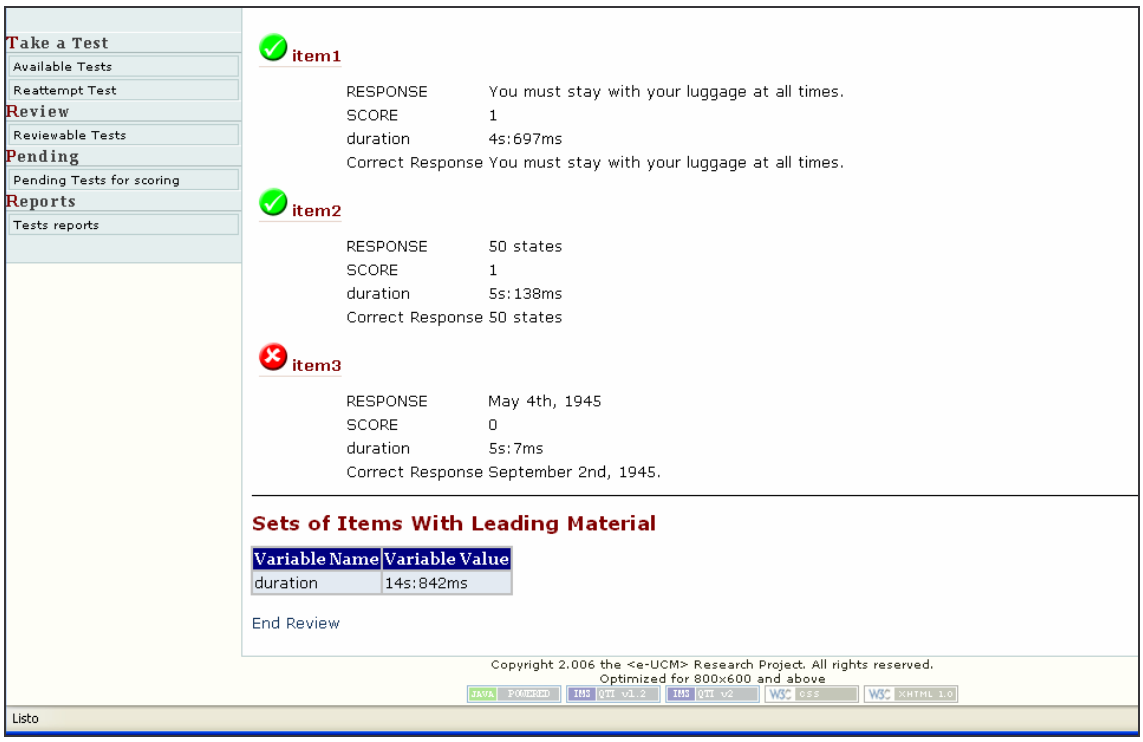


Ilustración 76: Pantalla de correción de test

15.1.4.2 Corrección de preguntas

Si el alumno ha elegido la realización de una pregunta únicamente, se le mostrará al alumno la pregunta seleccionada y una vez contestada se le mostrará al alumno una pantalla resumen con información sobre la pregunta y su resultado. Para salir de esta pantalla presionar el enlace “End Review”.

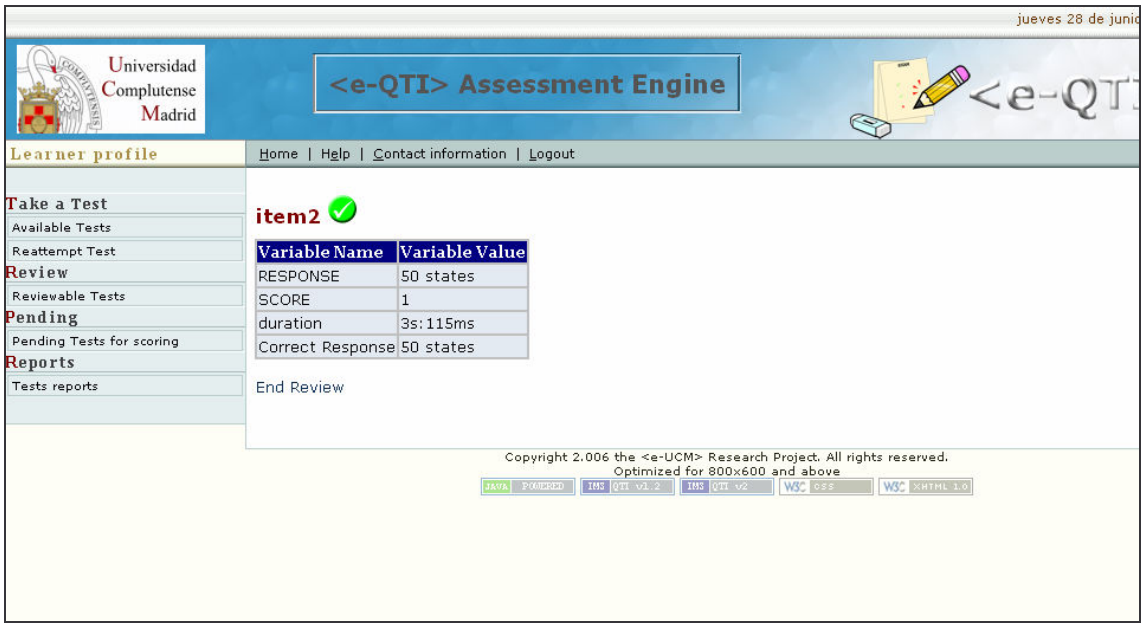


Ilustración 77: Pantalla de correción de preguntas

15.1.5 Como responder a las preguntas

15.1.5.1 Elección simple

Un alumno para contestar una pregunta de elección simple debe marcar solamente una de las posibles respuestas que se le ofrecen y presionar el botón "grade". A continuación se le mostrará al alumno la pantalla resumen de corrección de preguntas.

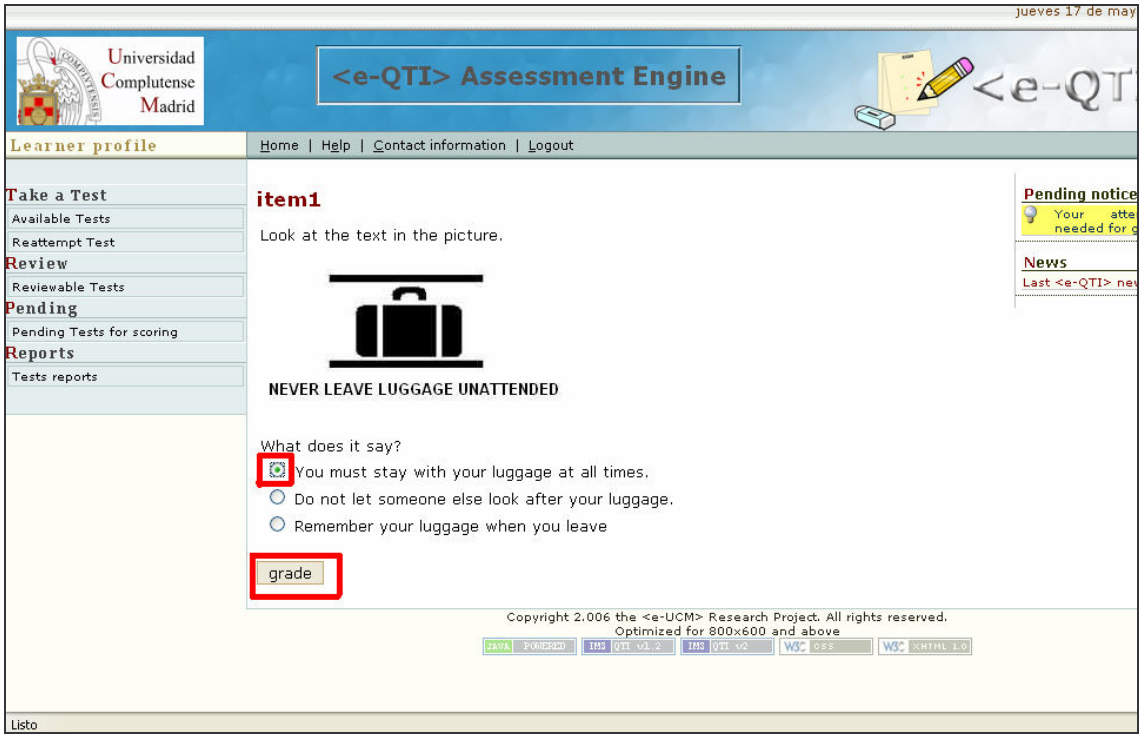


Ilustración 78: Elección Simple

15.1.5.2 Elección múltiple

Un alumno para contestar una pregunta de elección múltiple debe seleccionar todas las posibles respuestas que considere correctas y luego presionar el botón "grade". A continuación se le mostrará al alumno la pantalla resumen de corrección de preguntas.

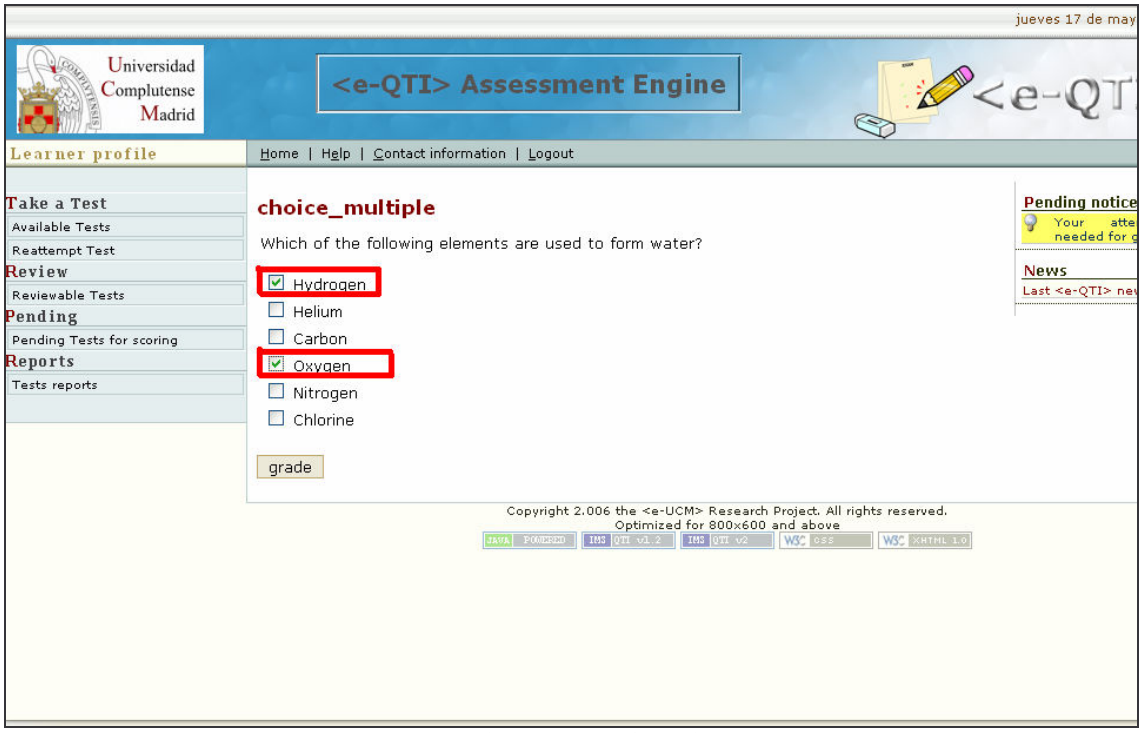


Ilustración 79: Elección Múltiple

15.1.5.3 Asociación

Un alumno para contestar una pregunta de asociación debe seleccionar en cada una de las relaciones mostradas el valor que considere correcto de las opciones que se le muestran en los menús para realizar las parejas correctas y luego presionar el botón "grade". A continuación se le mostrará al alumno la pantalla resumen de corrección de preguntas.

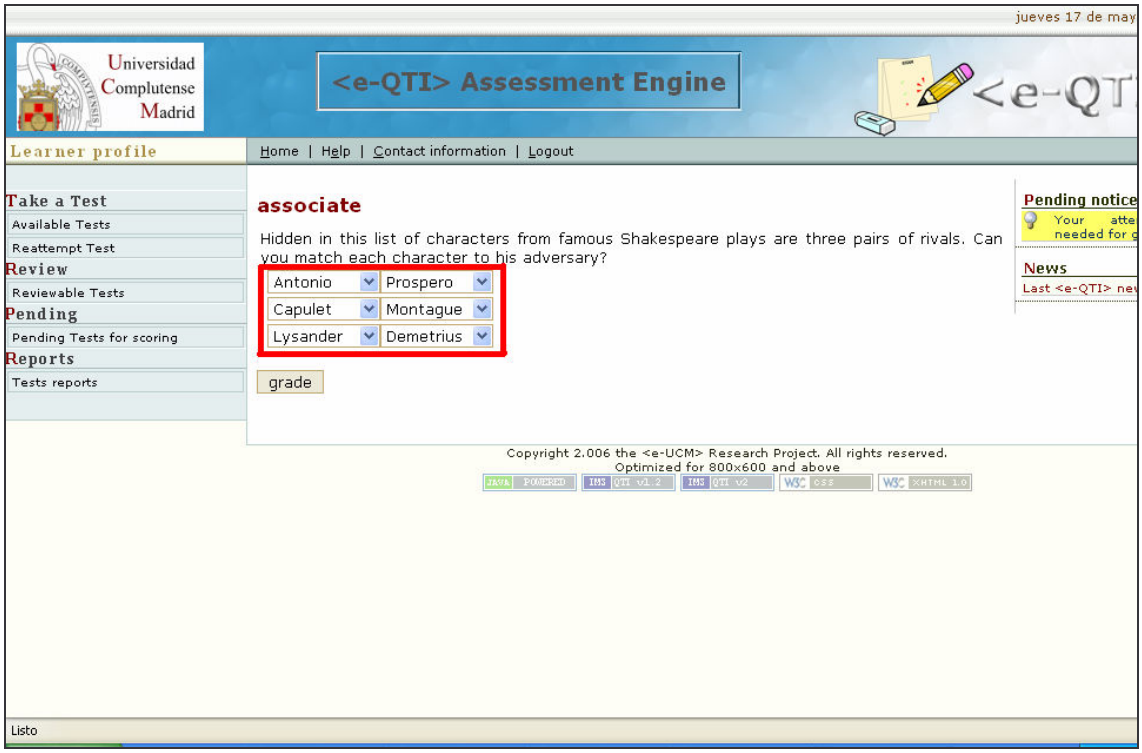


Ilustración 80: Asociación

15.1.5.4 Opciones en línea

Un alumno para contestar una pregunta de opciones en línea debe seleccionar la opción que considera correcta dentro del menú desplegable que se le muestra en esta pregunta y luego presionar el botón "grade". A continuación se le mostrará al alumno la pantalla resumen de corrección de preguntas.

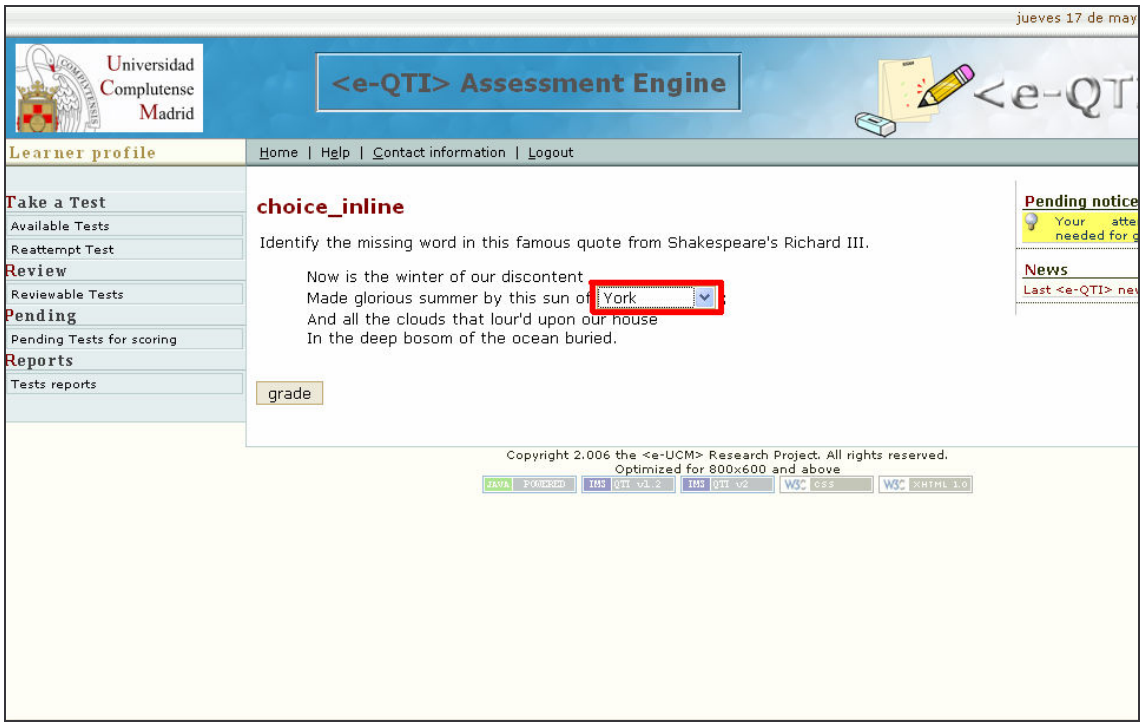


Ilustración 81: Opciones en Línea

15.1.5.5 Entrada de texto

Un alumno para contestar una pregunta de entrada de texto debe introducir la palabra o frase que considere oportuna en el hueco en blanco de la pregunta y luego presionar el botón "grade". A continuación se le mostrará al alumno la pantalla resumen de corrección de preguntas.

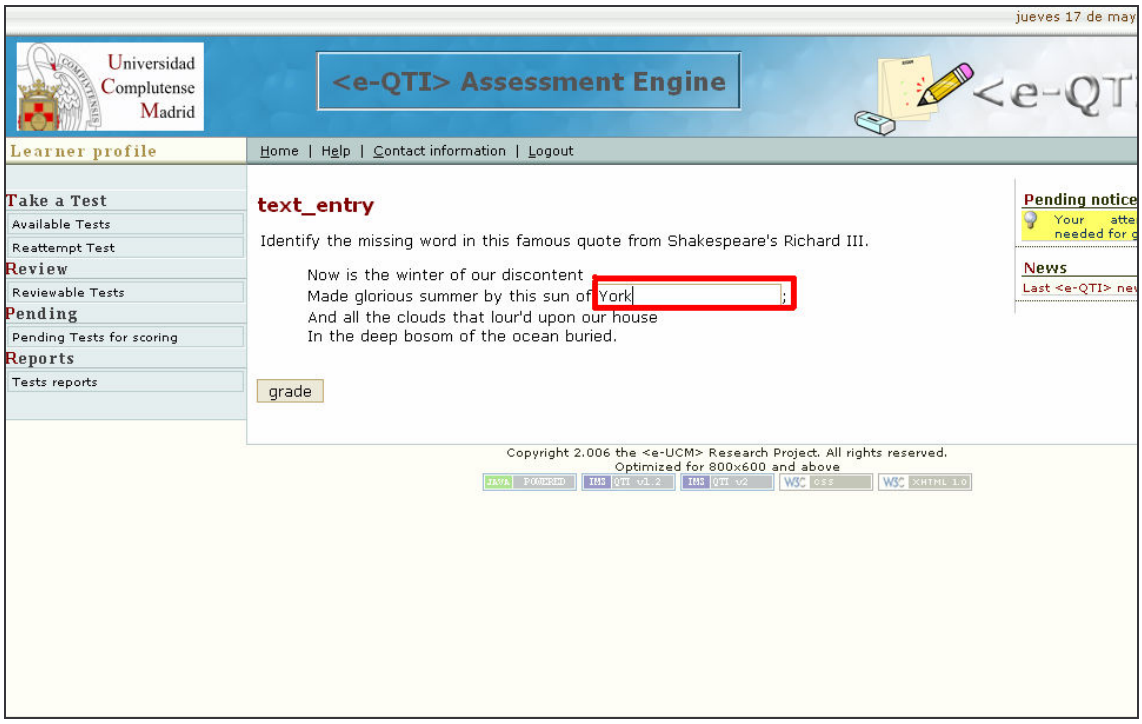


Ilustración 82: Entrada de texto

15.2 Manual de usuario del tutor

15.2.1 Pantalla de inicio de sesión

Para que un tutor utilice la aplicación debe estar dado de alta y registrado en el sistema con role de tutor, una vez que esta registrado el tutor debe de ir a la pantalla inicial del sistema donde debe introducir su usuario y su contraseña y a continuación presionar el botón de login.

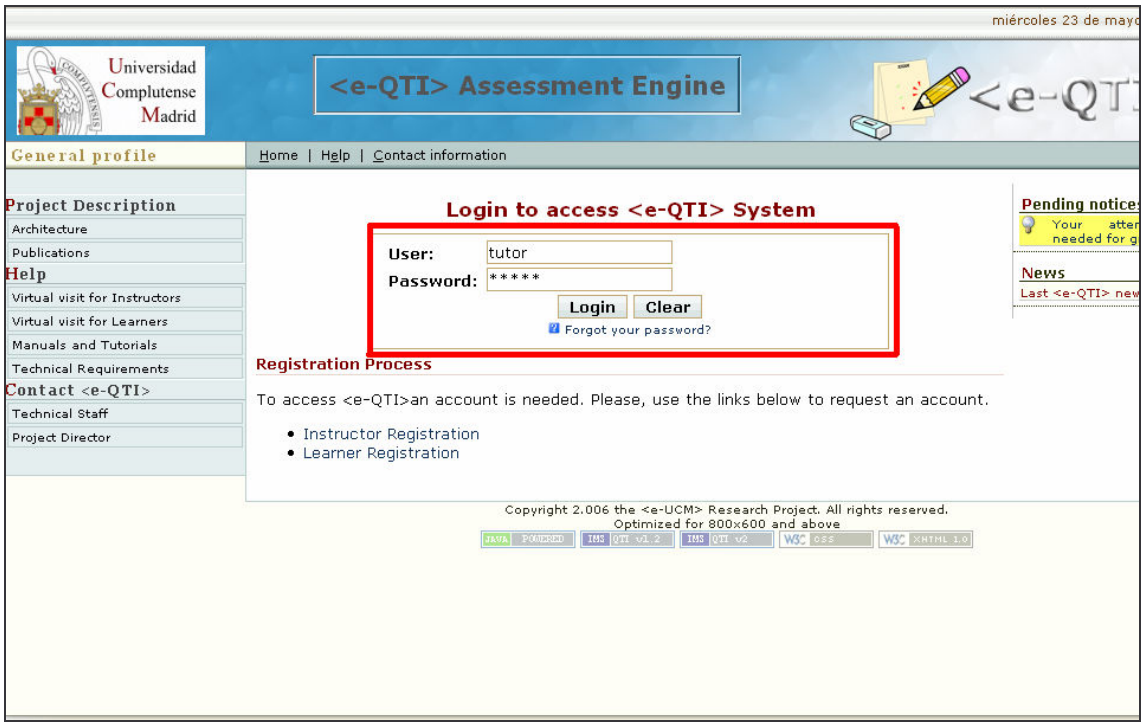


Ilustración 83: Inicio de sesión del tutor

15.2.2 Pantalla de elección de Site

La siguiente pantalla que se le muestra al tutor, es la pantalla donde se visualizan los "sites" o espacios habilitados para la creación y edición de los test y preguntas, cada "site" corresponderá a una asignatura o área temática y contendrá los test y preguntas creadas en ese "site". El tutor deberá elegir a que "site" quiere ir y presionar a su enlace.

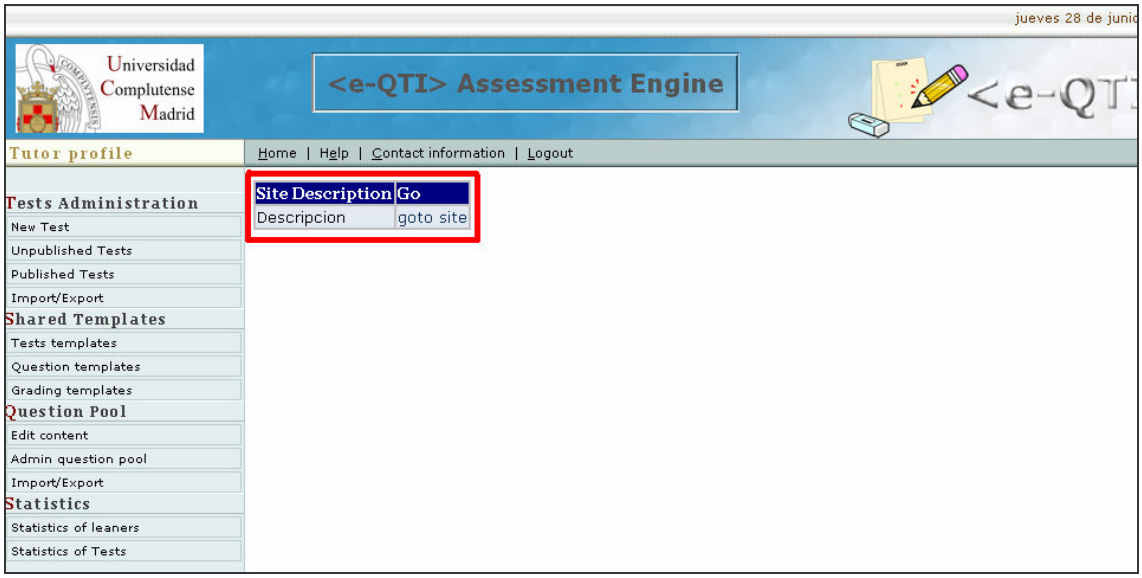


Ilustración 84: Elección del Site

15.2.3 Pantalla de creación y edición

Esta pantalla se divide en dos partes claramente diferenciadas, en la primera (marcada con el número 1 en la imagen) se muestran las preguntas y test creados por el usuario.
Si se presiona el enlace de cada uno de los ítems o test (El enlace se encuentra en la columna "Edit") se llevará al usuario a la pantalla de edición del item o test elegido. La parte segunda (marcada con el número 2), está compuesta únicamente por dos enlaces,"New Test" y "New Item". Si se presiona "New Test" se le mostrará al usuario la pantalla de creación de un nuevo test, si el usuario presiona el enlace de "New Item" se le mostrará al usuario la pantalla de creación de nuevas preguntas.

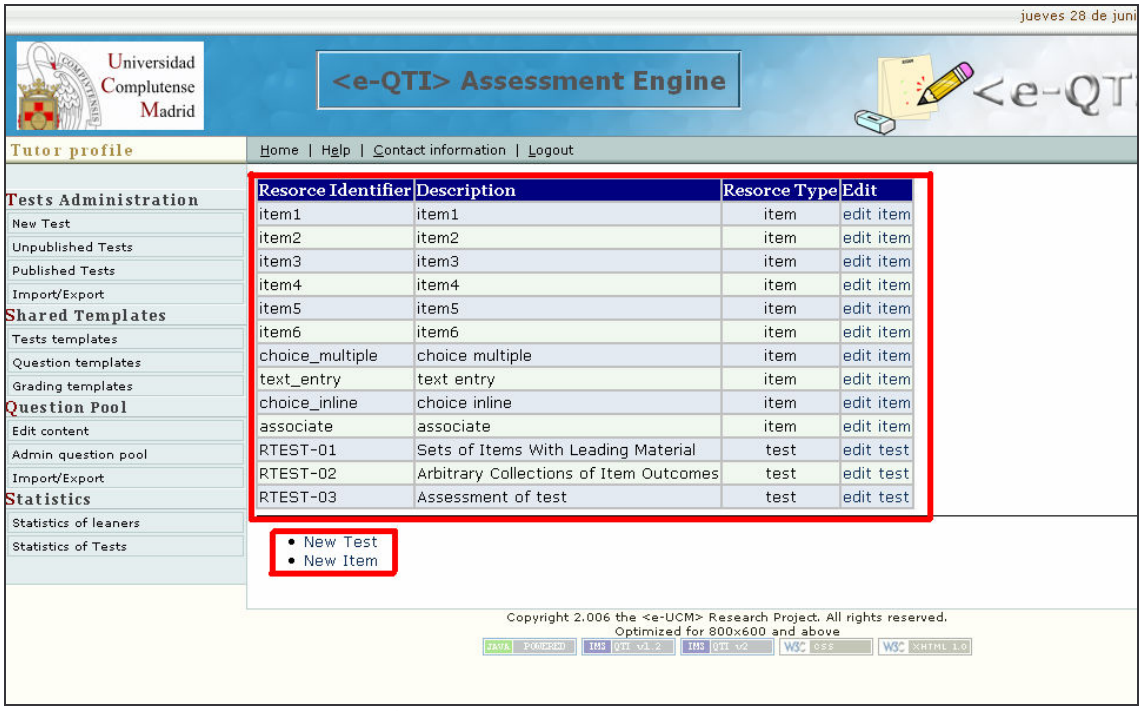


Ilustración 85: Pantalla de selección para el tutor

15.2.4 Creación

15.2.4.1 Pantalla de creación de un test

Esta es la pantalla en la que el tutor rellena los atributos generales del test. El principal atributo que se muestra en esta pantalla en es el "Title" o título del test, este es el único atributo que es obligatorio para todo test. El otro atributo que se puede rellenar en esta pantalla es "Description", este atributo es una breve descripción del test, el contenido de "Description" se muestra en la pantalla de creación y edición ya comentada. Por último una vez terminado de rellenar estos dos campos el tutor debe presionar el botón "Create" para confirmar la creación del test.

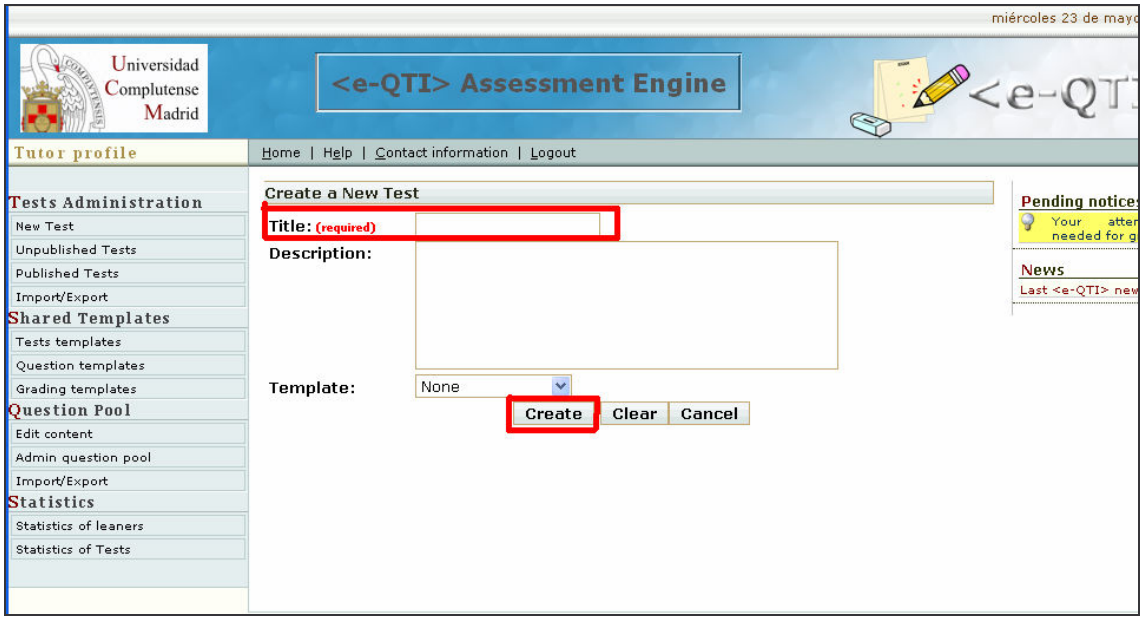


Ilustración 86: Creación de un test

Como podemos ver se nos muestra el test creado en la pantalla de creación y edición.

Ilustración 87: Mostrar el test creado

15.2.4.2 Creación de una pregunta

15.2.4.2.1 Pantalla elección del tipo de pregunta

En esta pantalla se muestran un listado con el tipo de preguntas que se pueden crear, actualmente se pueden editar seis tipos de preguntas:

- Verdadero / Falso (marcado como 1 en el dibujo)
- Elección Simple (2)
- Elección Múltiple (3)
- Entrada de texto (4)
- Opciones en línea (5)
- Asociación (6)

Para crear una pregunta del tipo elegido sólo se tiene que presionar sobre el nombre del tipo seleccionado.

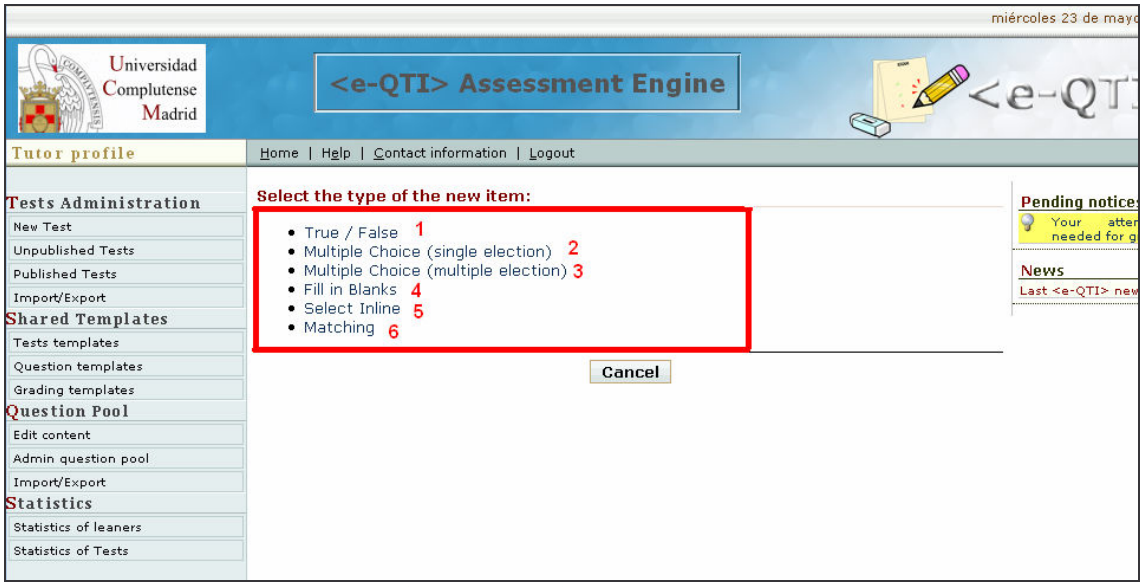


Ilustración 88: Selección de la pregunta a crear

15.2.4.2.2 Creación Verdadero/Falso

En esta pantalla el tutor puede crear una pregunta del tipo Verdadero/Falso. La pantalla esta divida en varias partes:

- Título: es el título de la pregunta (marcada con el número 1 en la imagen).
- Enunciado: Aquí el tutor debe escribir el enunciado y toda la información que se considere relevante sobre él (marcada con el número 2).
- Definición de las respuestas: En este apartado se le muestra una plantilla al tutor con los valores de las respuestas (Verdadero y Falso en nuestro caso), estos valores pueden ser editados según las preferencias del tutor.
- Selección de respuesta: Aquí el tutor deberá seleccionar cual de las dos respuestas que hay es la correcta.

Una vez que se haya rellenado los campos se debe presionar el botón "Create".

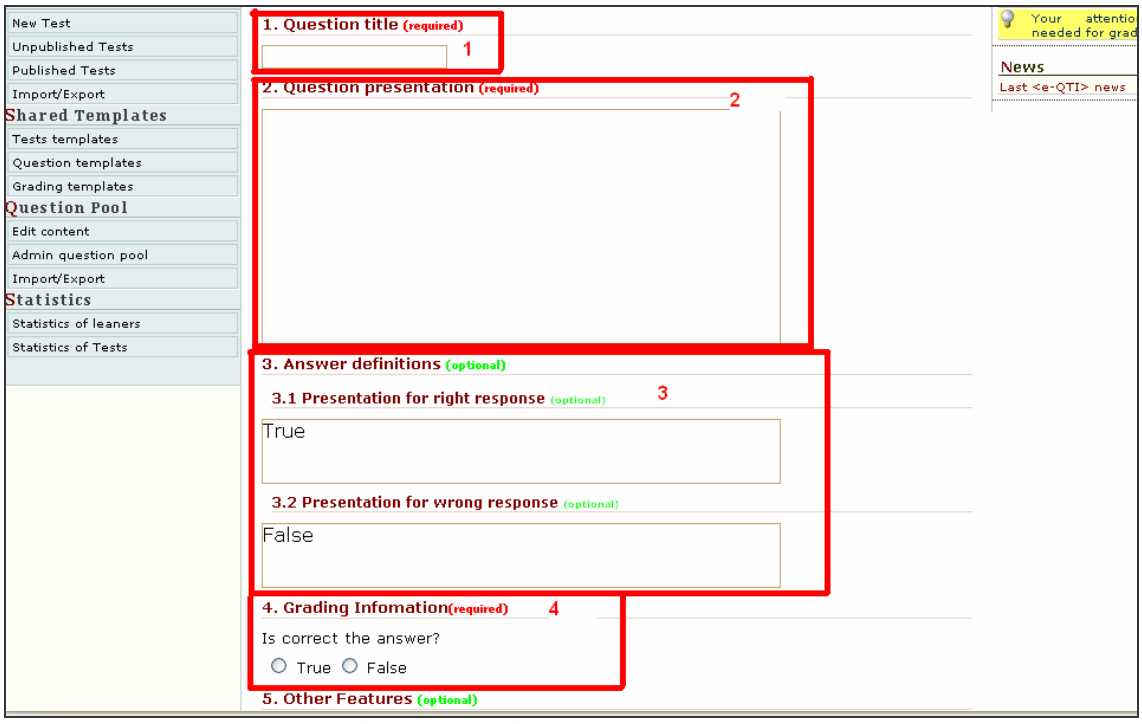


Ilustración 89: Creación de preguntas Verdadero/Falso

A continuación se nos mostrará en la pantalla de creación y edición la pregunta nueva.

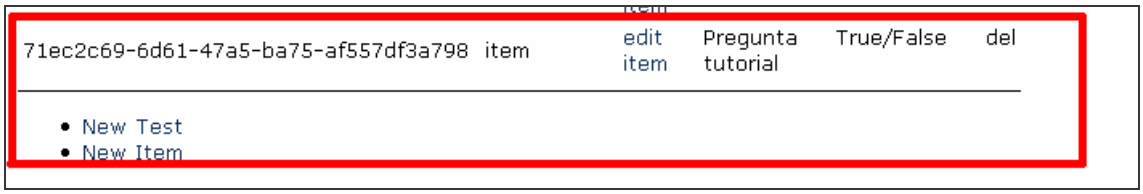


Ilustración 90: Mostrar la pregunta True/False creada

15.2.4.2.3 Creación Elección Simple

En esta pantalla el tutor puede crear una pregunta del tipo Elección Simple. La pantalla esta divida en varias partes:

- Título: es el título de la pregunta (marcada con el número 1 en la imagen).
- Enunciado: Aquí el tutor debe escribir el enunciado y toda la información que se considere relevante sobre él (marcada con el número 2).
- Definición de las respuestas: En este apartado el tutor debe añadir las respuestas de su pregunta (Marcado con el número 3). Esta parte será explicada más detenidamente posteriormente.
- Selección de respuesta: Aquí el tutor deberá seleccionar cual de las respuestas que hay es la correcta.

Una vez que se haya rellenado los campos se debe presionar el botón "Create".

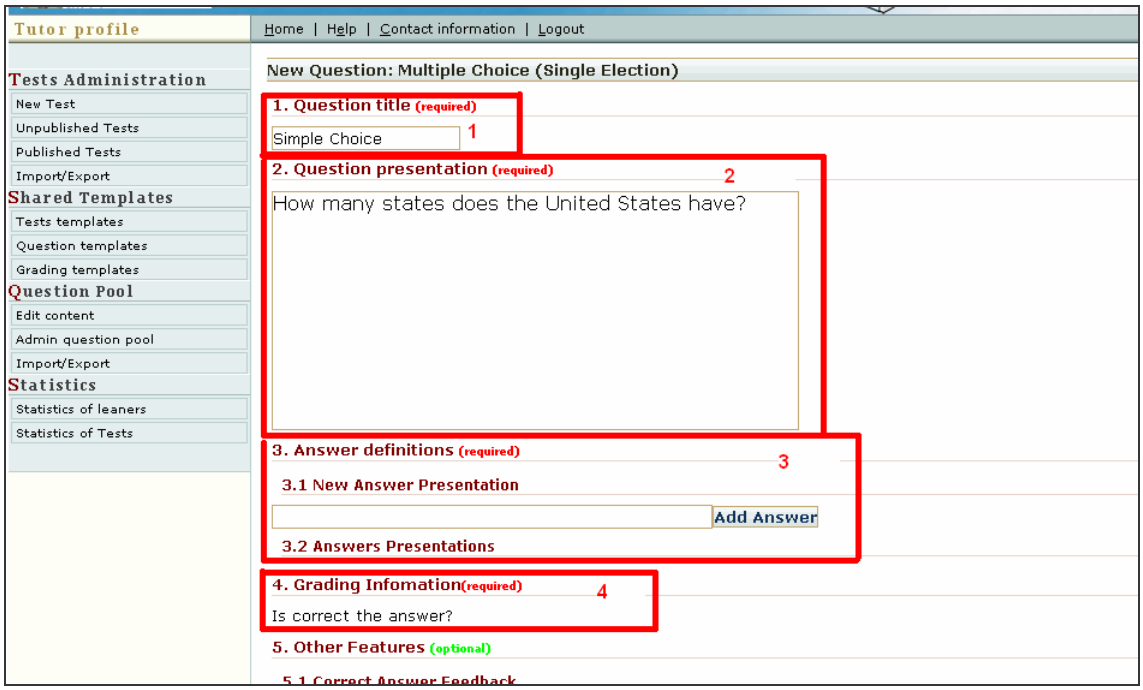


Ilustración 91: Creación de la pregunta Elección Simple

15.2.4.2.3.1 Añadir res7puestas

Para añadir la primera respuesta el tutor debe rellenar el campo marcado en la siguiente figura y a continuación presionar el botón “Add Answer”.

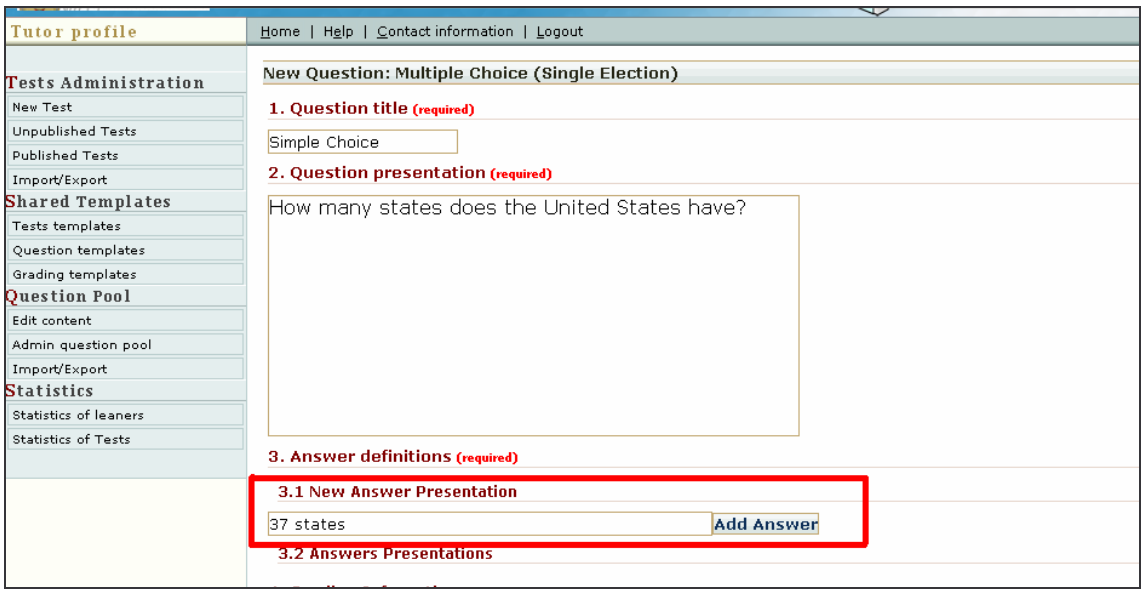


Ilustración 92: Creación de la primera respuesta en Elección Simple

Una vez añadida una respuesta se nos mostrará en el apartado “3.1 Answer Presentation”. La nueva respuesta añadida también nos aparecerá en el apartado de “Selección de respuestas”.

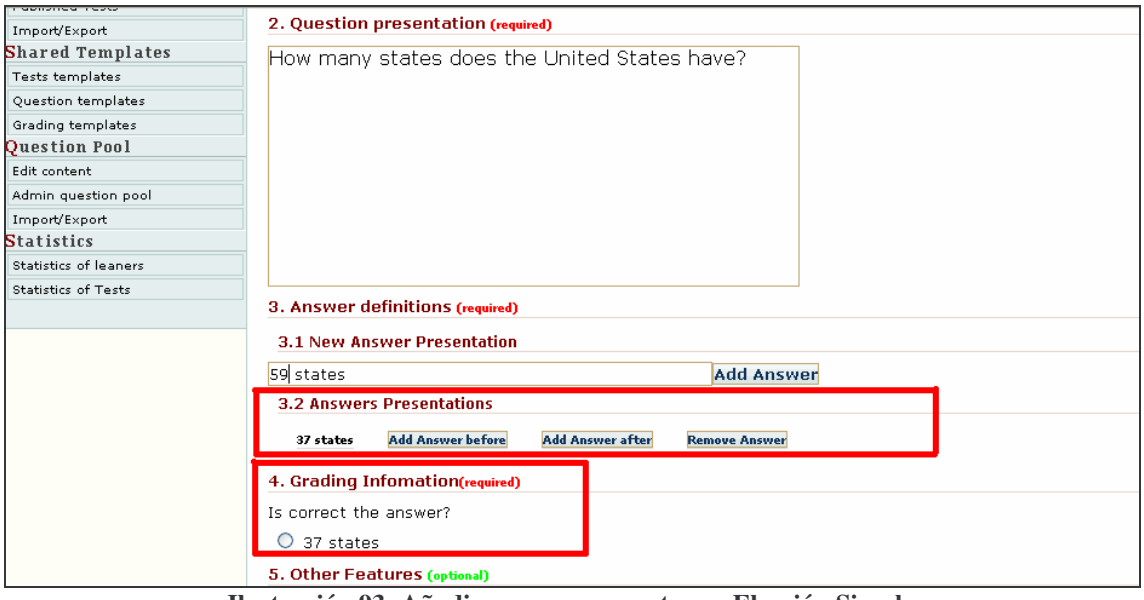


Ilustración 93: Añadir nuevas respuestas en Elección Simple

Una vez que nuestra pregunta tiene una primera respuesta se nos habilitan en el apartado “3.2 Answer Presentations” tres nuevos botones:

- Add Answer befote: este botón sirve para añadir la nueva respuesta antes de la respuesta en la cual hemos presionado el botón.
- Add Answer alter: este botón sirve para añadir la nueva respuesta después de la respuesta en la cual hemos presionado el botón.
- Remove Answer: Sirve para borrar una respuesta creada.

Por último, una vez que introducidas todas las respuestas deseadas, el tutor debe marcar cual es la respuesta correcta.

Ahora aparecerá en la pantalla de creación y edición nuestra nueva pregunta.

Ilustración 94: Mostrar la nueva pregunta de Elección Simple Creada

15.2.4.2.4 Creación Elección Múltiple

En esta pantalla el tutor puede crear una pregunta del tipo Elección Simple. La pantalla esta divida en varias partes:

- Título: es el título de la pregunta (marcada con el número 1 en la imagen).
- Enunciado: Aquí el tutor debe escribir el enunciado y toda la información que se considere relevante sobre él (marcada con el número 2).
- Definición de las respuestas: En este apartado el tutor debe añadir las respuestas de su pregunta (Marcado con el número 3). Esta parte será explicada más detenidamente posteriormente.
- Información de corrección: En este apartado el tutor debe seleccionar cual es la puntuación máxima y mínima de una pregunta y cual son sus respuestas correctas (Marcado con el número 4).

Una vez que se haya rellenado los campos se debe presionar el botón “Create”.

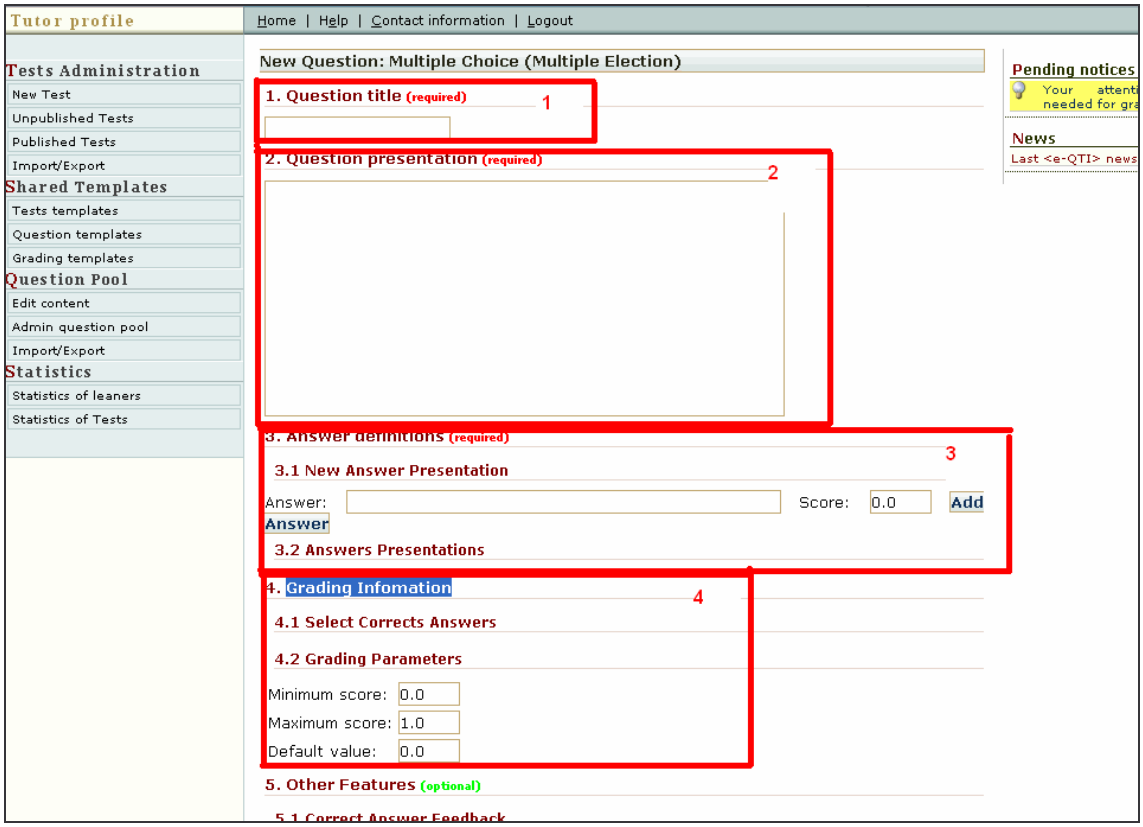


Ilustración 95: Creación de Multiple Choice

15.2.4.2.4.1 Añadir respuestas

Para añadir cada respuesta el tutor debe rellenar los dos campos marcados en la figura. En el primero de ellos se debe rellenar con el texto de la respuesta. El segundo campo es el valor que se le otorgará al alumno en caso de que responda con dicha respuesta. Si se esta editando la primera pregunta se debe presionar el botón "Add Answer".

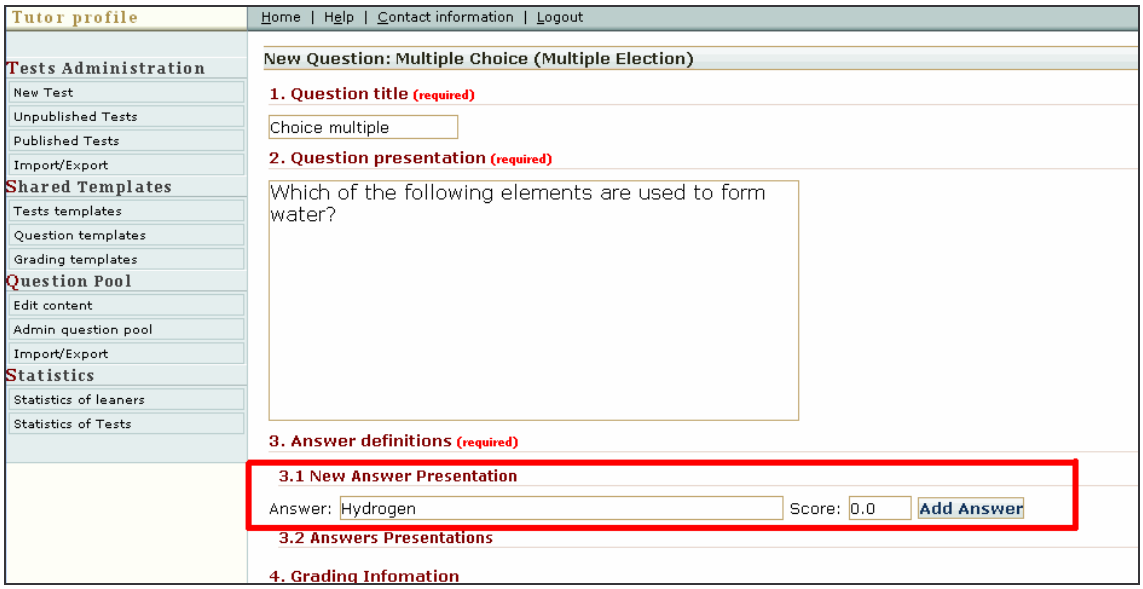


Ilustración 96: Añadir la primera respuesta en Multiple Choice

Una vez que nuestra pregunta tiene una primera respuesta se nos habilitan en el apartado "3.2 Answer Presentations" tres nuevos botones:

- Add Answer befote: este botón sirve para añadir la nueva respuesta antes de la respuesta en la cual hemos presionado el botón.
- Add Answer alter: este botón sirve para añadir la nueva respuesta después de la respuesta en la cual hemos presionado el botón.
- Remove Answer: Sirve para borrar una respuesta creada.

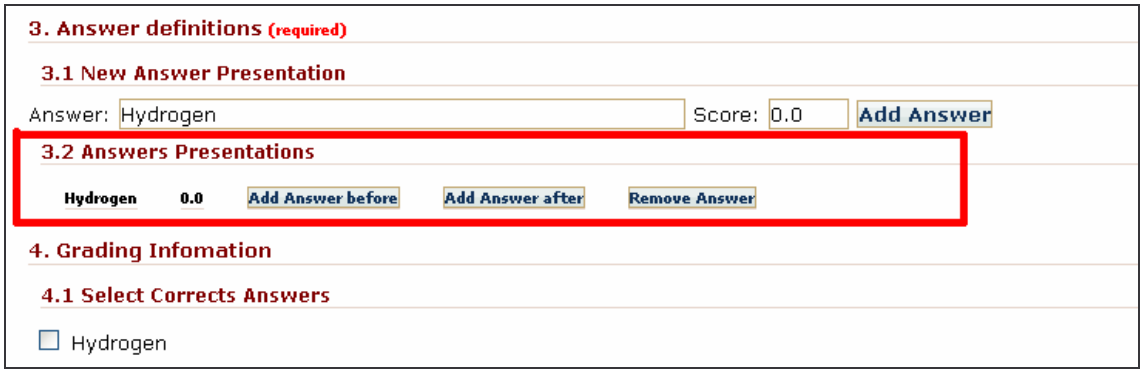


Ilustración 97: Añadir respuestas en Multiple Choice

Por último una vez que hemos añadido todas las respuestas posibles debemos marcar cuales de estas respuestas son correctas, para ello debemos dirigirnos al apartado "4.1 Select Correct Answer" y marcarlas. Una vez marcadas las respuestas correctas debemos rellenar el apartado "4.2 Grading Parameters" y rellenar los campos: Mínima Puntuación, Máxima Puntuación y Puntuación por Defecto con los valores deseados para cada uno de los campos comentados.

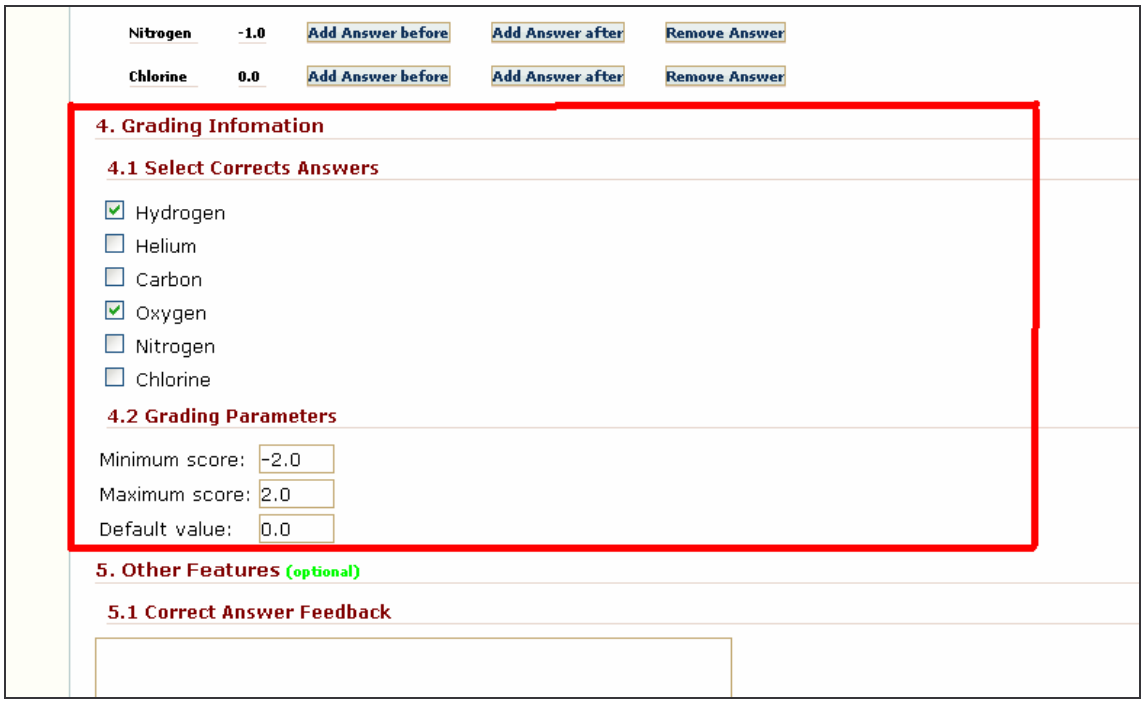


Ilustración 98: Selección de las respuestas correctas en Multiple Choice

15.2.4.2.5 Creación de Entrada de Texto

En esta pantalla el tutor puede crear una pregunta del tipo Entrada de Texto. La pantalla esta divida en varias partes:

- Título: es el título de la pregunta (marcada con el número 1 en la imagen).
- Enunciado: Aquí el tutor debe escribir el enunciado y toda la información que se considere relevante sobre él (marcada con el número 2). El tutor debe marcar con "[]" el lugar donde quiere que aparezca el espacio en blanco para que el alumno introduzca el texto de respuesta.
- Definición de las respuestas: En este apartado el tutor debe añadir las respuestas de su pregunta (Marcado con el número 3). Esta parte será explicada más detenidamente posteriormente.
- Información de corrección: En este apartado el tutor debe seleccionar cual es la puntuación máxima y mínima de una pregunta y cual son sus respuestas correctas (Marcado con el número 4).

Una vez que se haya rellenado los campos se debe presionar el botón "Create".

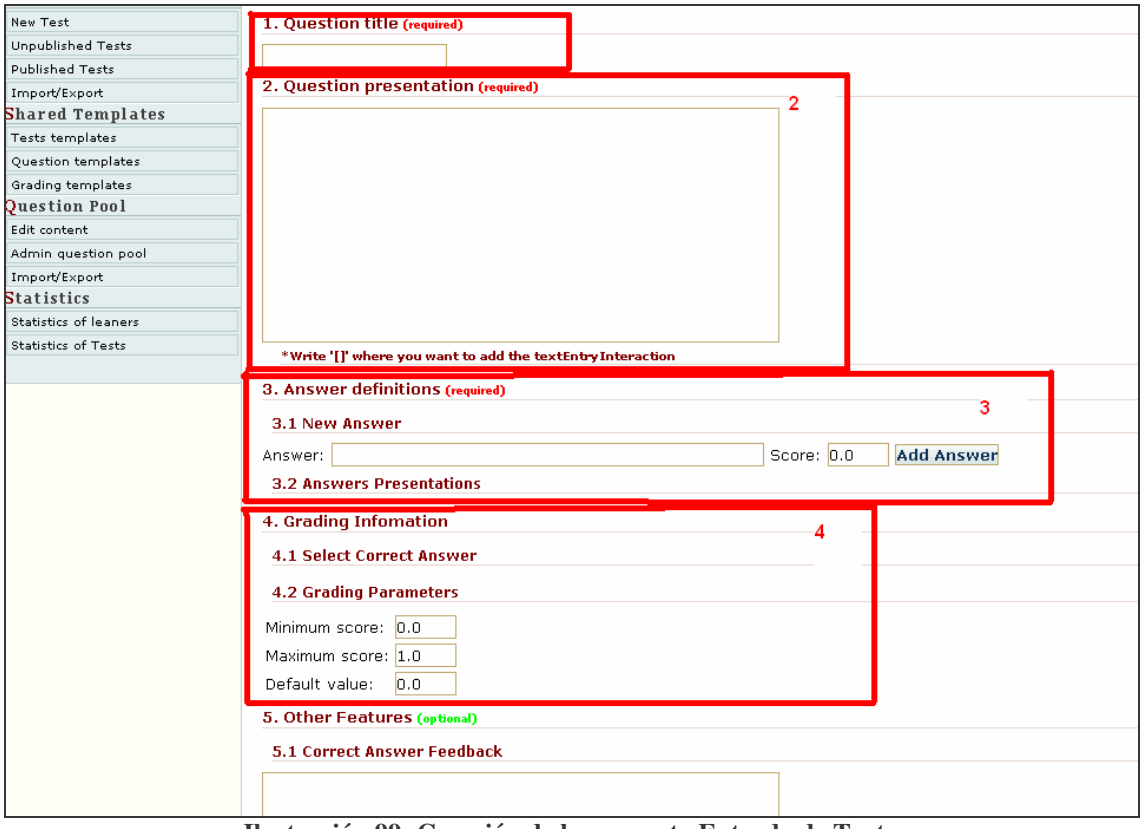


Ilustración 99: Creación de la pregunta Entrada de Texto

15.2.4.2.5.1 Añadir respuestas

Para añadir cada respuesta el tutor debe rellenar los dos campos marcados en la figura. En el primero de ellos se debe rellenar con el texto de la respuesta. El segundo campo es el valor de la respuesta, positivo si es correcta y negativo si es falsa. Si se esta editando la primera pregunta se debe presionar el botón "Add Answer".

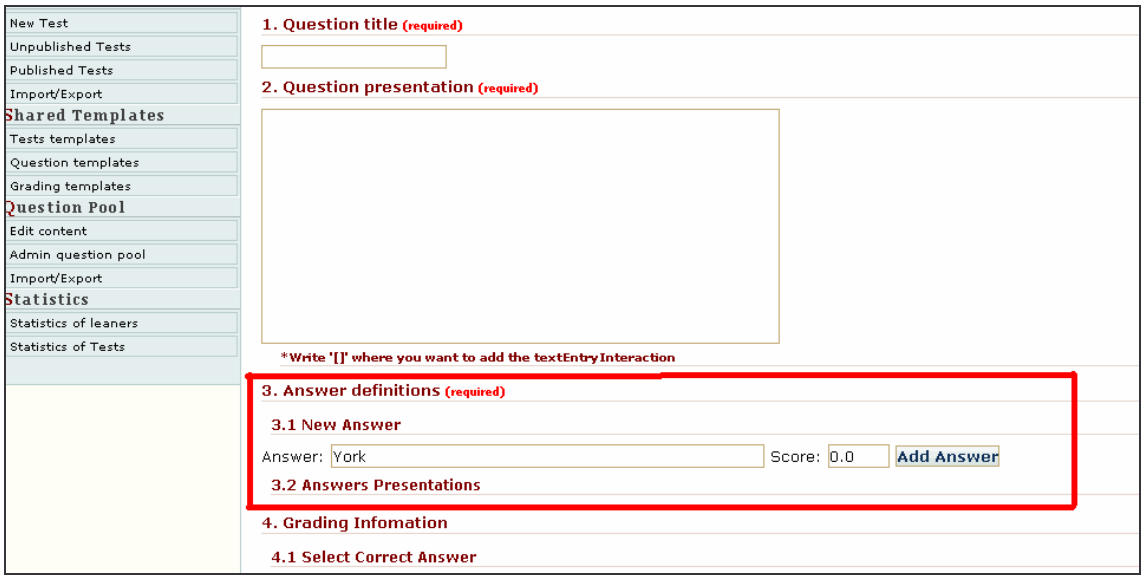


Ilustración 100: Añadir la primera respuesta en Entrada de Texto

Una vez que nuestra pregunta tiene una primera respuesta se nos habilitan en el apartado "3.2 Answer Presentations" tres nuevos botones:

- Add Answer befote: este botón sirve para añadir la nueva respuesta antes de la respuesta en la cual hemos presionado el botón.
- Add Answer alter: este botón sirve para añadir la nueva respuesta después de la respuesta en la cual hemos presionado el botón.
- Remove Answer: Sirve para borrar una respuesta creada.

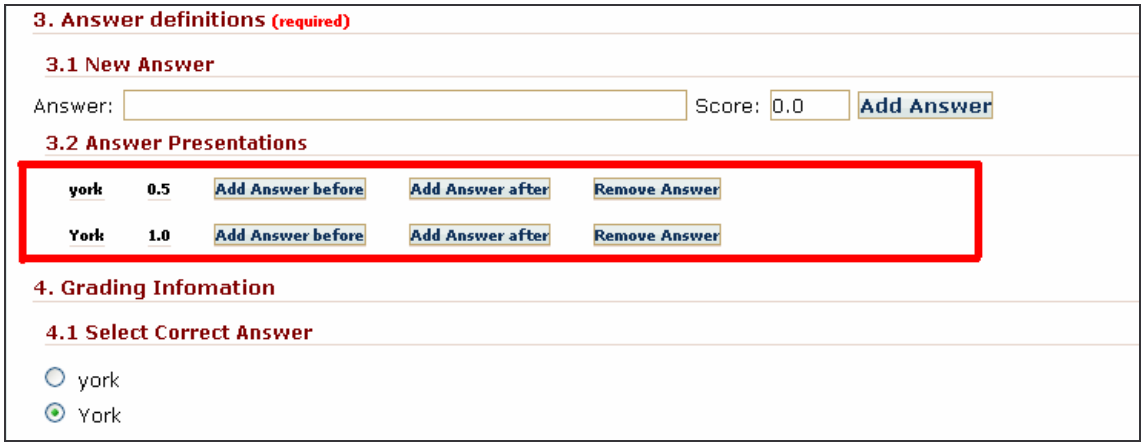


Ilustración 101: Añadir respuestas en Entrada de Texto

Por último una vez que hemos añadido todas las respuestas posibles debemos marcar cual de estas respuestas es la correcta, para ello debemos dirigirnos al apartado "4.1 Select Correct Answer" y marcarla. Una vez marcada la respuesta correcta debemos rellenar el apartado "4.2 Grading Parameters" y rellenar los campos: Mínima Puntuación, Máxima Puntuación y Puntuación por Defecto con los valores deseados para cada uno de los campos comentados

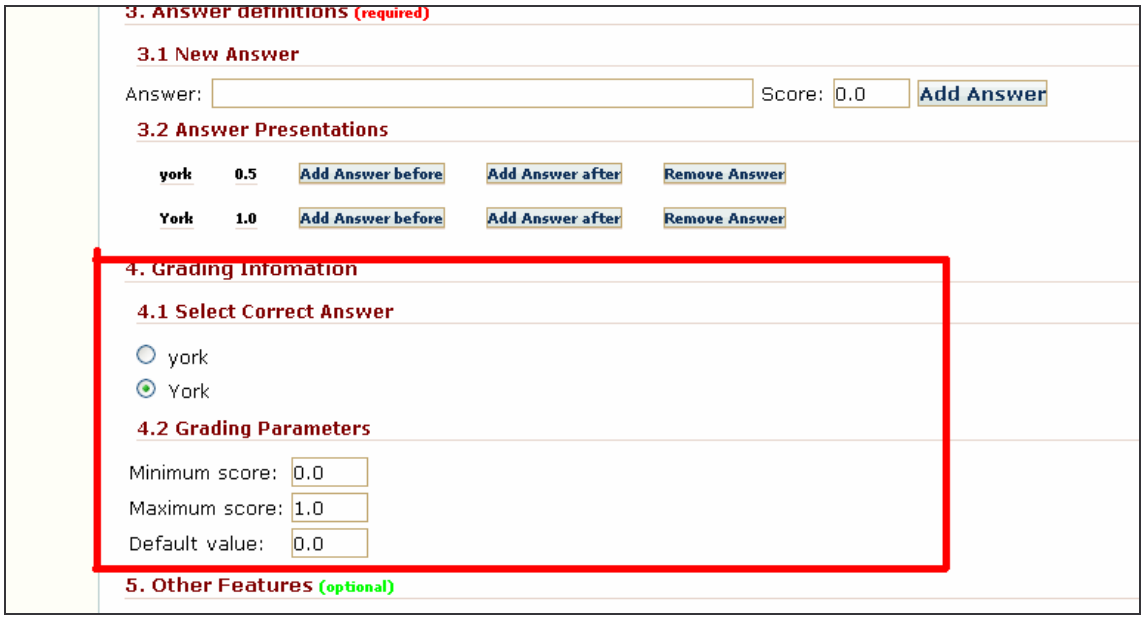


Ilustración 102: Selección de la respuesta correcta en Entrada de texto

15.2.4.2.6 Creación de Selección en Línea

En esta pantalla el tutor puede crear una pregunta del tipo Selección el Línea. La pantalla esta divida en varias partes:

- Título: es el título de la pregunta (marcada con el número 1 en la imagen).
- Enunciado: Aquí el tutor debe escribir el enunciado y toda la información que se considere relevante sobre él (marcada con el número 2). El tutor debe marcar con "[]" el lugar donde quiere que aparezca el espacio en blanco para que el alumno introduzca el texto de respuesta.
- Definición de las respuestas: En este apartado el tutor debe añadir las respuestas de su pregunta (Marcado con el número 3). Esta parte será explicada más detenidamente posteriormente.
- Información de corrección: En este apartado el tutor debe seleccionar cual es la puntuación máxima y mínima de una pregunta y cual son sus respuestas correctas (Marcado con el número 4).

Una vez que se haya rellenado los campos se debe presionar el botón "Create".

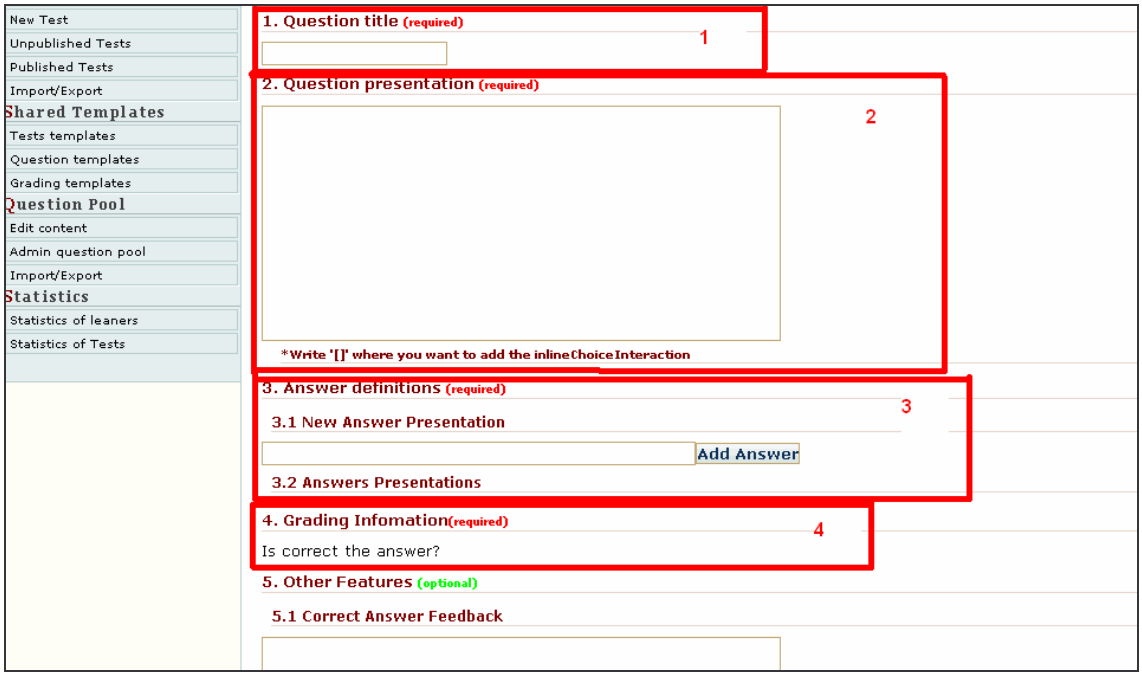


Ilustración 103: Creación de Selección en Línea

15.2.4.2.6.1 Añadir respuestas

Para añadir cada respuesta el tutor debe rellenar los dos campos marcados en la figura. En el primero de ellos se debe rellenar con el texto de la respuesta. El segundo campo es el valor de la respuesta, positivo si es correcta y negativo si es falsa. Si se esta editando la primera pregunta se debe presionar el botón "Add Answer".

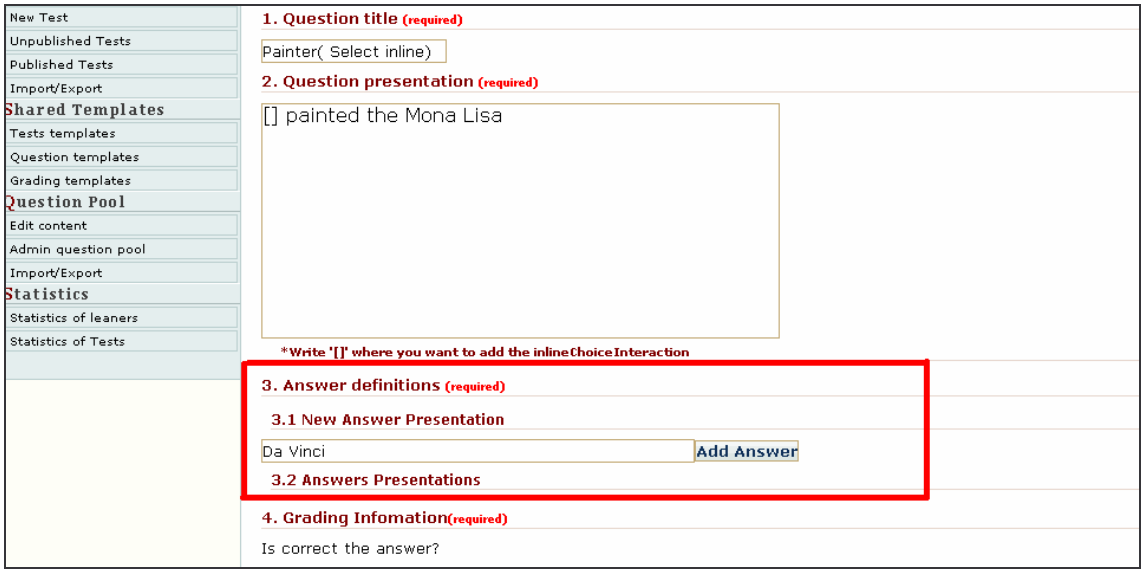


Ilustración 104: Añadir la primera respuesta en Selección en Línea

Una vez que nuestra pregunta tiene una primera respuesta se nos habilitan en el apartado "3.2 Answer Presentations" tres nuevos botones:

- Add Answer befote: este botón sirve para añadir la nueva respuesta antes de la respuesta en la cual hemos presionado el botón.
- Add Answer alter: este botón sirve para añadir la nueva respuesta después de la respuesta en la cual hemos presionado el botón.
- Remove Answer: Sirve para borrar una respuesta creada.

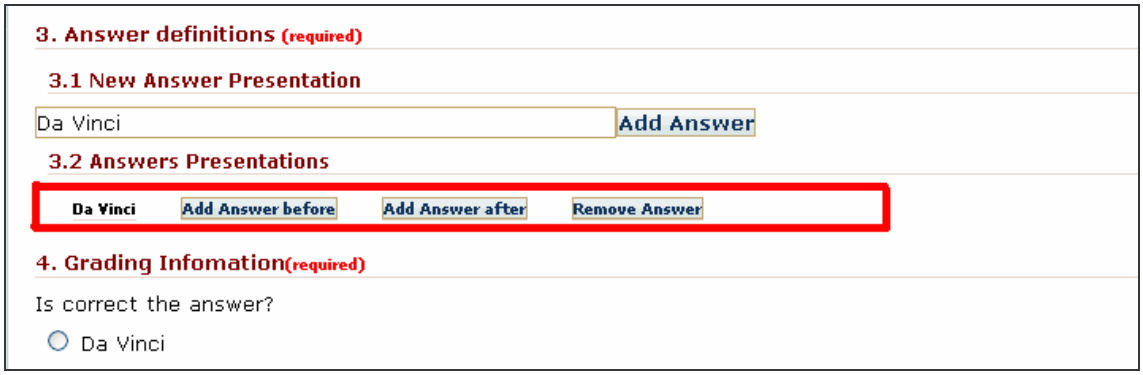


Ilustración 105: Añadir respuestas en Selección en ínea

Por último una vez que hemos añadido todas las respuestas posibles debemos marcar cual de estas respuestas es la correcta, para ello debemos dirigirnos al apartado "4.1 Select Correct Answer" y marcarla.

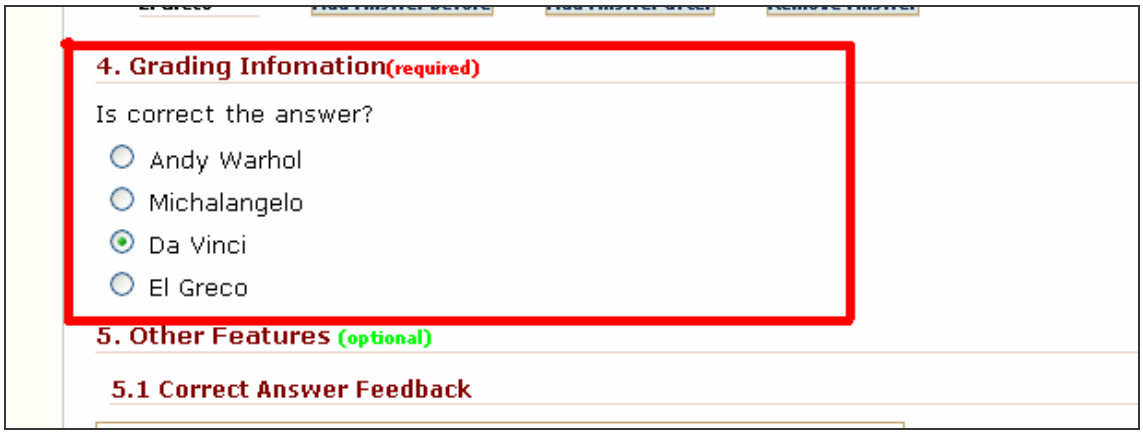


Ilustración 106: Selección de la respuesta correcta en Select Inline

15.2.4.2.7 Creación de Asociación

En esta pantalla el tutor puede crear una pregunta del tipo Selección el Línea. La pantalla esta divida en varias partes:

- Título: es el título de la pregunta (marcada con el número 1 en la imagen).
- Enunciado: Aquí el tutor debe escribir el enunciado y toda la información que se considere relevante sobre él (marcada con el número 2).
- Definición de las respuestas: En este apartado el tutor debe añadir las respuestas de su pregunta (Marcado con el número 3). Esta parte será explicada más detenidamente posteriormente.

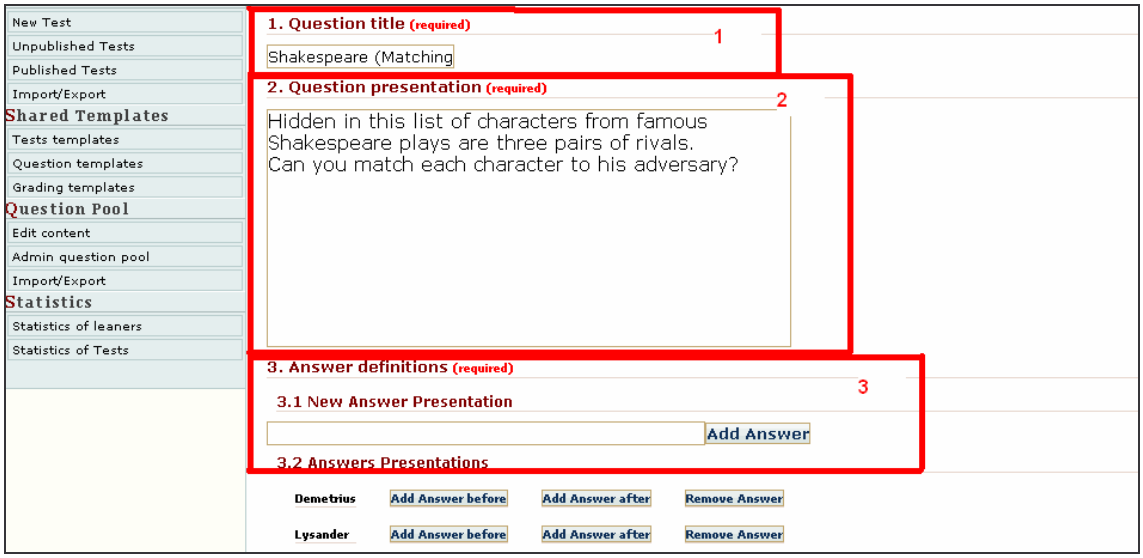


Ilustración 107: Creación de la pregunta Asociación 1

- Información de corrección: En este apartado el tutor debe seleccionar cual es la puntuación máxima y mínima de una pregunta y cual son sus respuestas correctas (Marcado con el número 4).

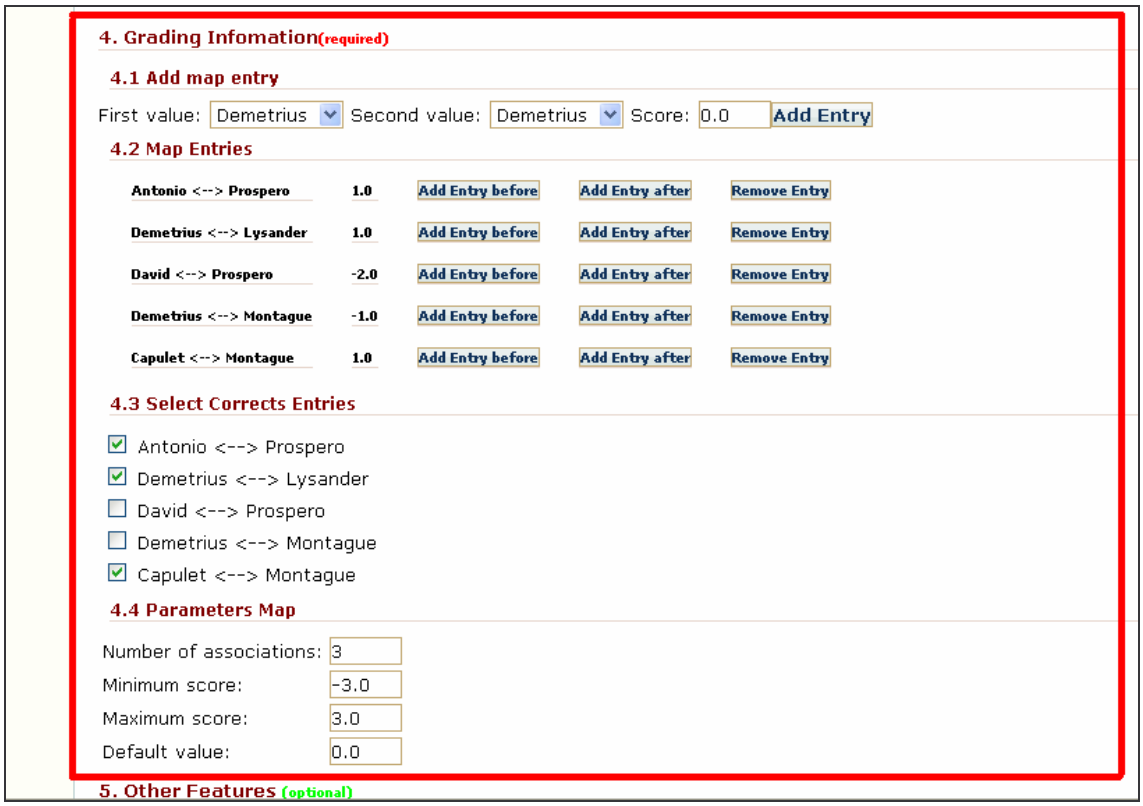


Ilustración 108: Creación de la pregunta Asociación 2

15.2.4.2.7.1 Añadir respuestas

Para añadir cada respuesta lo primero que debe hacer el tutor es crear la lista a relacionar. Para crear un elemento de la relación debe rellenar el campo marcado en la figura. En el primero de ellos se debe rellenar con el texto de la respuesta. Si se esta editando la primera pregunta se debe presionar el botón “Add Answer”.

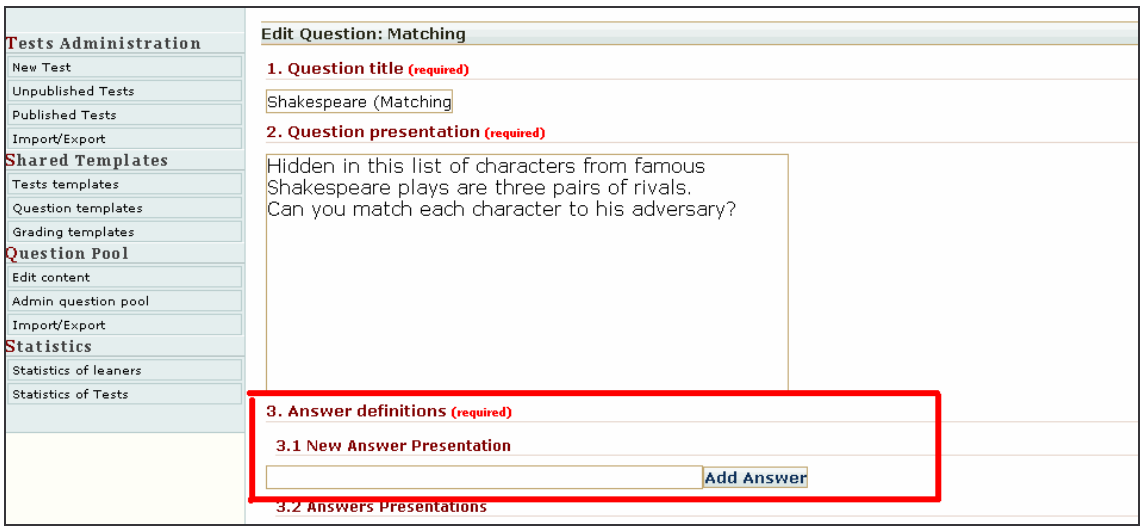


Ilustración 109: Añadir la primera opción en Asociación

Una vez que nuestra pregunta tiene una primera respuesta se nos habilitan en el apartado “3.2 Answer Presentations” tres nuevos botones:

- Add Answer befote: este botón sirve para añadir la nueva respuesta antes de la respuesta en la cual hemos presionado el botón.
- Add Answer alter: este botón sirve para añadir la nueva respuesta después de la respuesta en la cual hemos presionado el botón.
- Remove Answer: Sirve para borrar una respuesta creada.

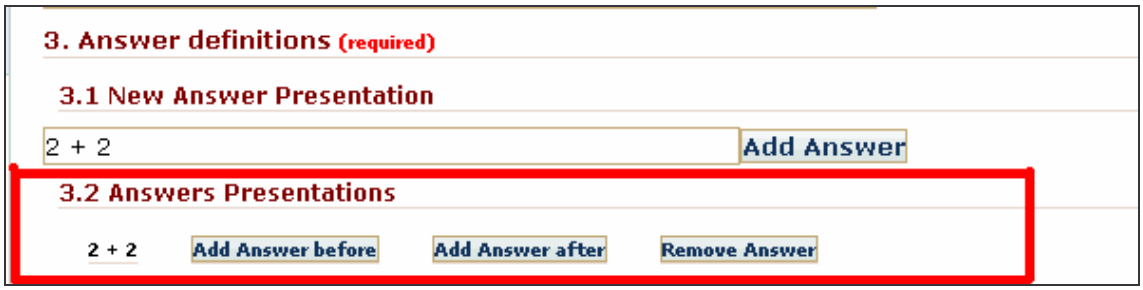


Ilustración 110: Añadir opciones en Asociación

Por último una vez que hemos añadido la lista a relacionar debemos crear las asociaciones. Para ello en el apartado marcado en la imagen seleccionamos un valor primero y un valor segundo y le asignamos un peso a esta asociación, este peso será la puntuación que se le asignará al alumno en caso de que responda con esta asociación. Para crear esta asociación sólo nos queda presionar el botón "Add Entry".

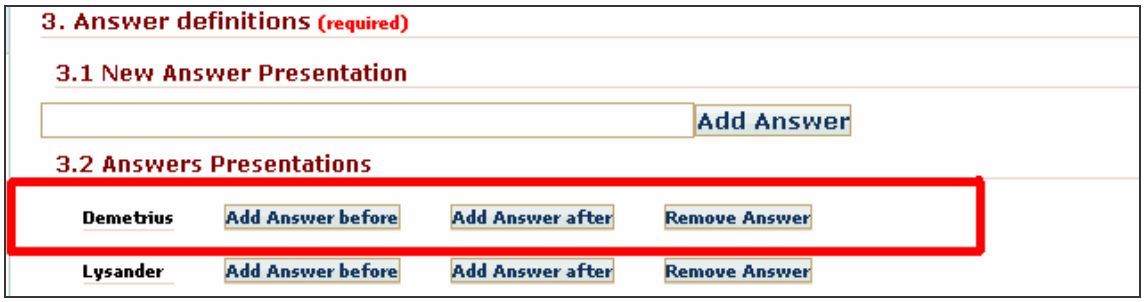


Ilustración 111: Añadir relaciones en la pregunta Asociación

Por último una vez que hemos añadido todas las respuestas posibles debemos marcar cuales de estas respuestas son correctas, para ello debemos dirigirnos al apartado "4.1 Select Correct Answer" y marcarlas. Una vez marcadas las respuestas correctas debemos rellenar el apartado "4.2 Grading Parameters" y rellenar los campos: Número de Asociaciones, Mínima Puntuación, Máxima Puntuación y Puntuación por Defecto con los valores deseados para cada uno de los campos comentados. El campo Número de Asociaciones indica el número de asociaciones que puede realizar el alumno.

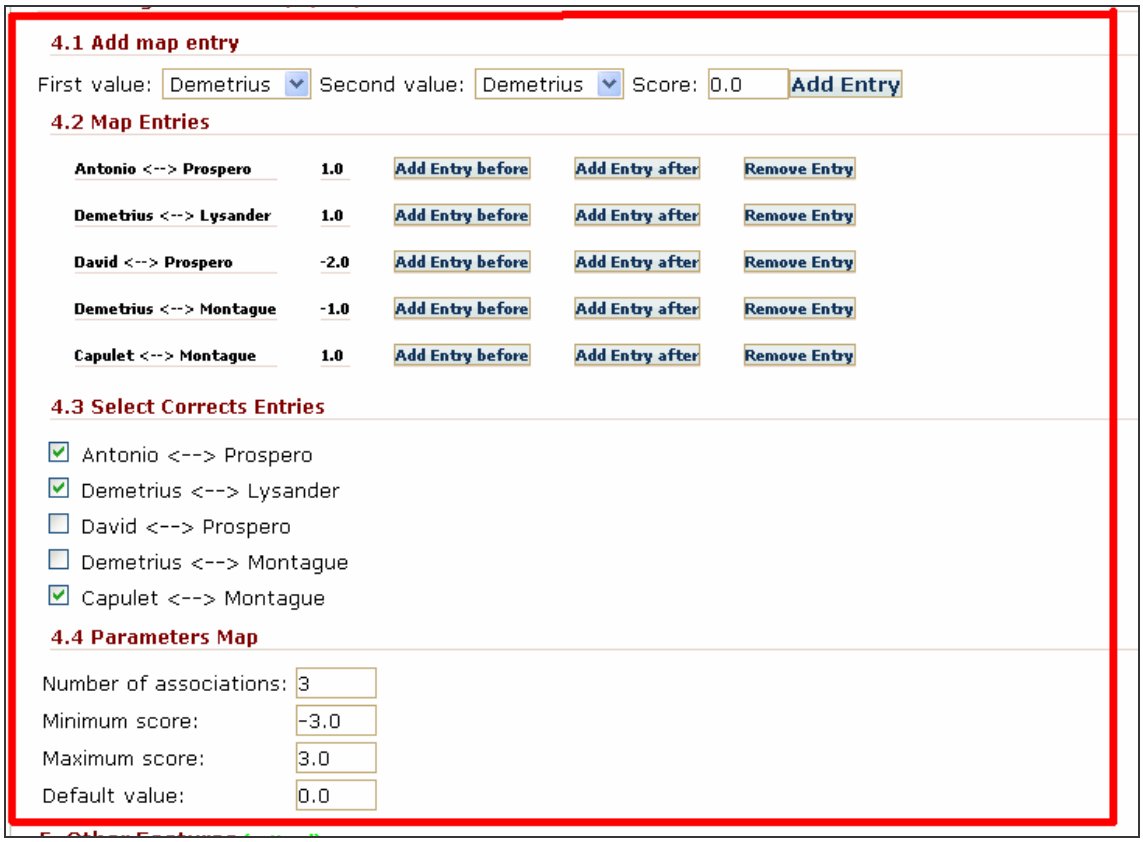


Ilustración 112: Selección de las respuestas correctas en Asociación

15.2.5 Edición

15.2.5.1 Edición de un test

Para editar un test debemos presionar su enlace (recuadrado en rojo y con nombre “edit test”) desde la pantalla de edicción y creacción.

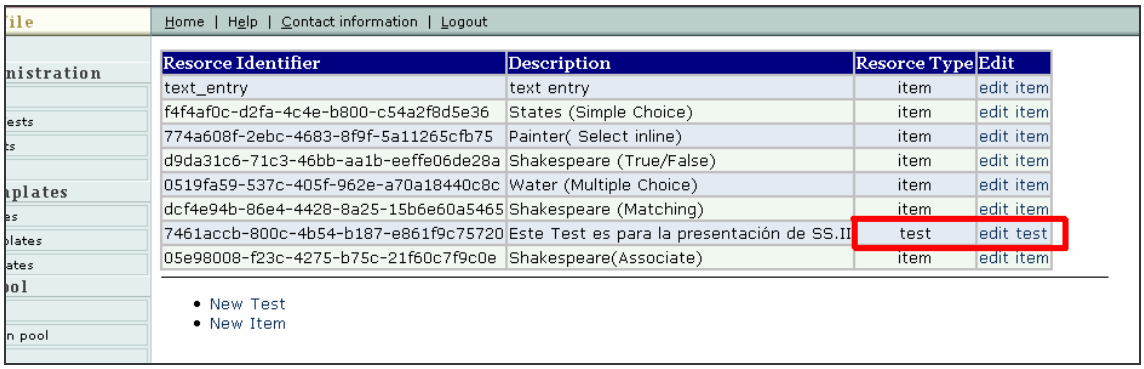


Ilustración 113 Elección del test a editar

Como se puede ver en la imagen en la parte superior de la pantallase nos mostrará el título del test y su descripción, los cuales habremos introducido en la pantalla de creacción del test, esta parte del test no puede ser editada.

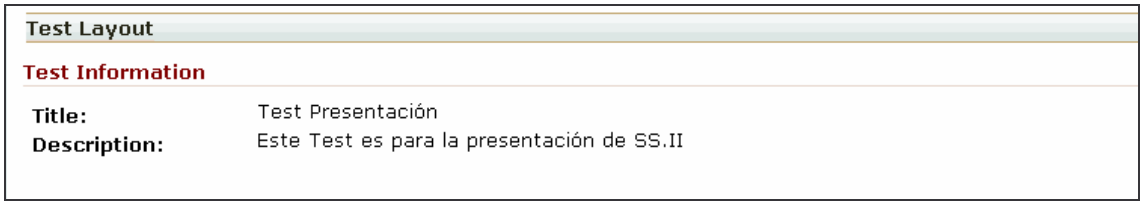


Ilustración 114 Mostrar Título y descripción en la edición

En la parte central de la pantalla nos aparecera un botón “Add Part” en el caso de que el test que hayamos editado no tenga una estructura ya creada. Este botón servira para añadir el primer “Test Part” de nuestro test. Cabe recordar que según el estandar QTI todo test se debe dividir en Test Part.

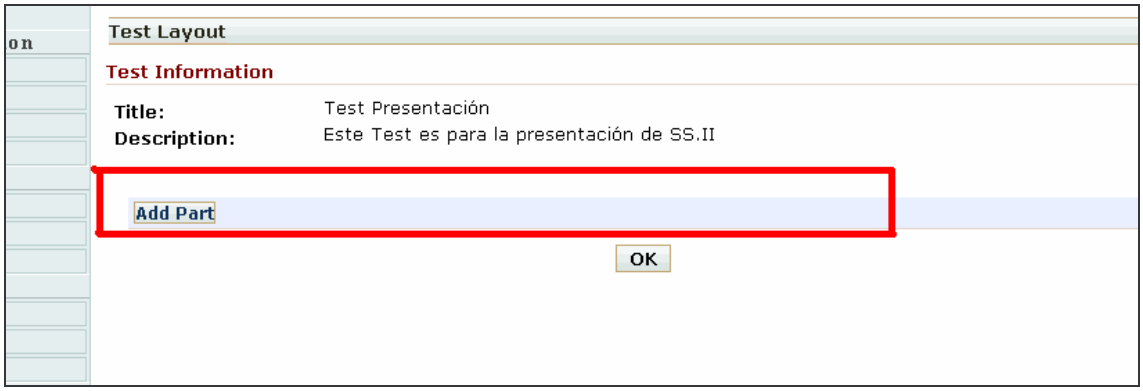


Ilustración 115 Añadir Test Part

Si estamos editando un test con una estructura ya creada nos aparecerá la estructura al completo.

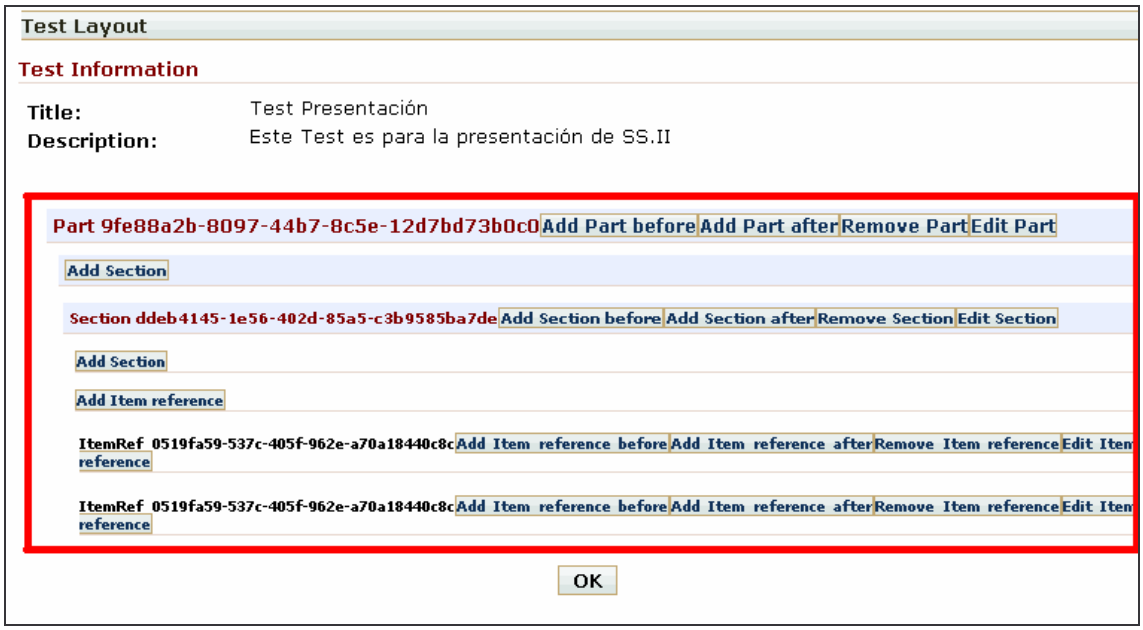


Ilustración 116 Mostrar estructura del test

Una vez creado el primer Test Part nos aparecerá en la parte central 4 nuevos botones para interactuar con dicho elemento:

- Add Part Before: Añadirá un nuevo Test Part antes del actual.
- Add Part After: Añadirá un nuevo Test Part después del actual.
- Remove Part: Borrará el Test Part actual y toda su estructura interna.
- Edit Part: Actualmente este botón no realiza ninguna acción.
- Add Section: Añade una nueva section al Test Part actual (todo Test Part debe tener al menos una section).

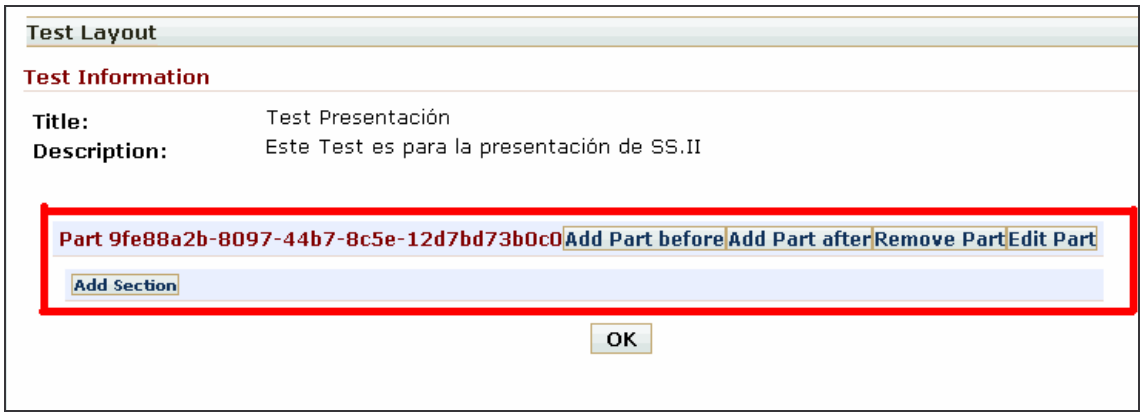


Ilustración 117 Botones del Test Part

Si nuestra estructura tiene alguna section, se nos mostrará a su vez seis nuevos botones:

- Add Section Before: Añade una nueva section antes de la actual.
- Add Section After: Añade una nueva section después de la actual.
- Remove Section: Elimina la section actual.
- Edit Section: Actualmente este botón no hace nada.
- Add Section: Añade una section dentro de la section actual (según el estandar QTI una section puede estar contenida dentro de otra section).
- Add Item Referente: presionando este botón se pueden incluir las preguntas a nuestra sección.

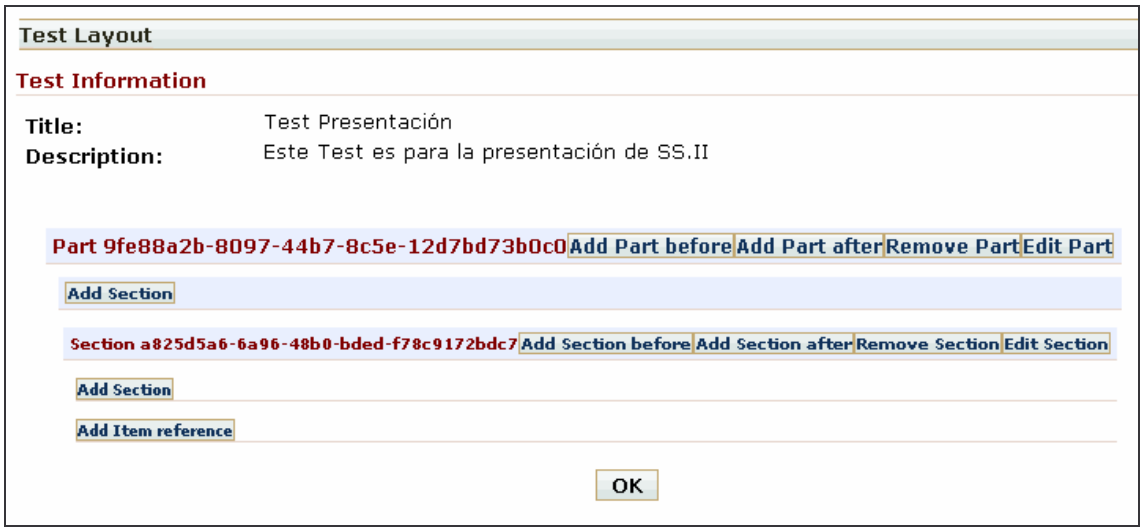


Ilustración 118 Botones de Section

Si presionamos el botón de "Add Item Referente" se nos redigirá a una pantalla en la que se nos mostrará las preguntas disponibles que podemos añadir a nuestro test. Para añadir una pregunta de las mostradas sólo hay que presionar el enlace "add" de la pregunta seleccionada (recuadrado en rojo en la figura).

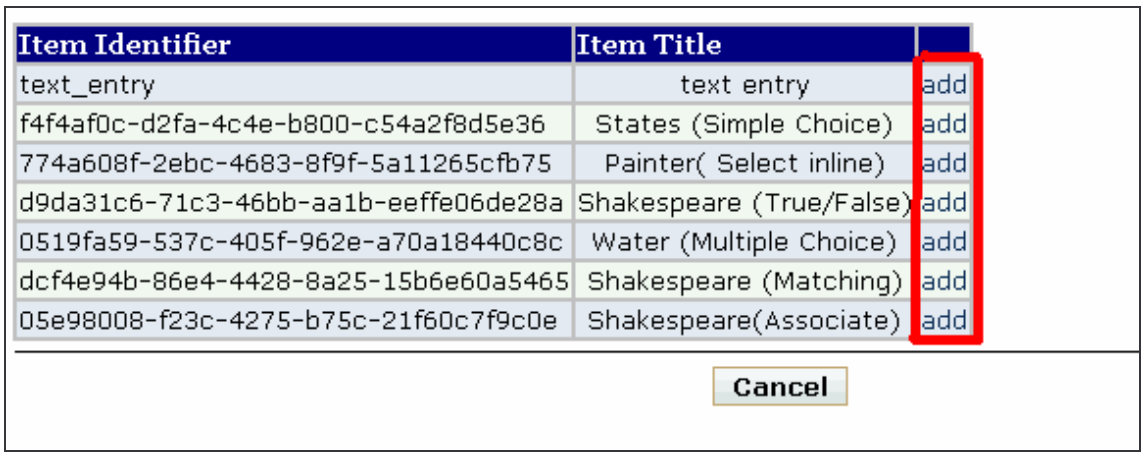


Ilustración 119 Recursos para añadir al test

Por último una vez editada la estructura de nuestro test sólo nos queda presionar el botón de “Ok” para guardar los cambios.

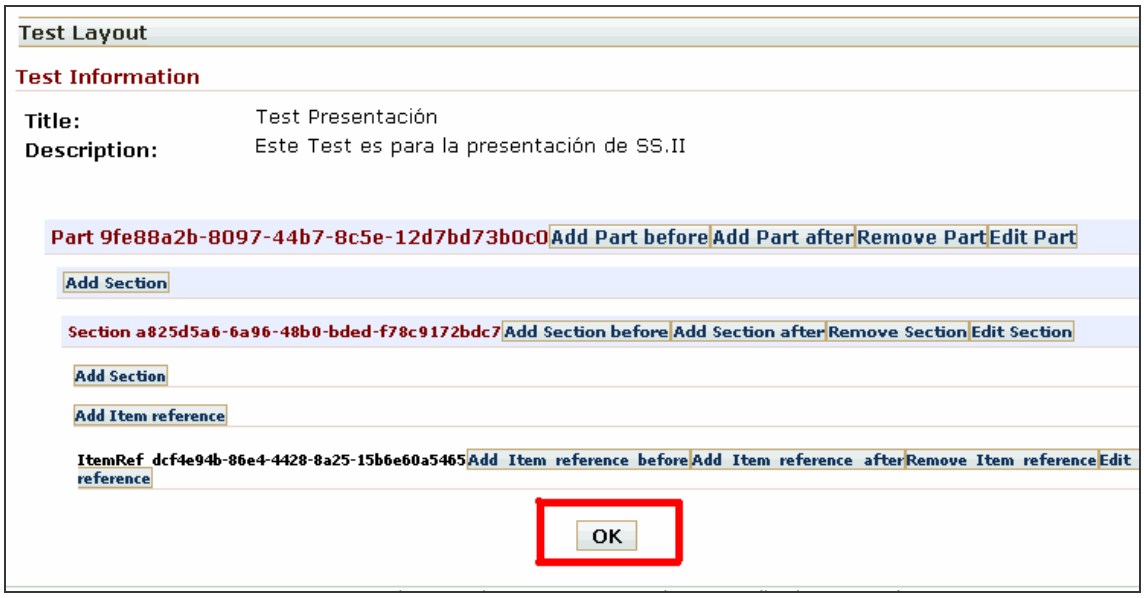
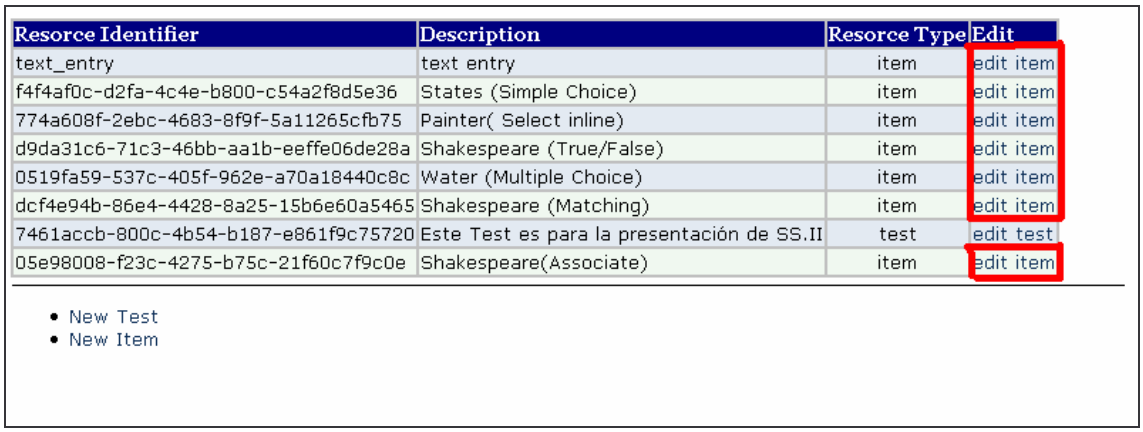


Ilustración 120 Fin de la edición del test

Edición de una pregunta

Para editar una pregunta debemos presionar el enlace “edit Item” de la pregunta correspondiente que queremos editar en la pantalla de edición y creación.



Resorce Identifier	Description	Resorce Type	Edit
text_entry	text entry	item	edit item
f4f4af0c-d2fa-4c4e-b800-c54a2f8d5e36	States (Simple Choice)	item	edit item
774a608f-2ebc-4683-8f9f-5a11265cfb75	Painter(Select inline)	item	edit item
d9da31c6-71c3-46bb-aa1b-eeffe06de28a	Shakespeare (True/False)	item	edit item
0519fa59-537c-405f-962e-a70a18440c8c	Water (Multiple Choice)	item	edit item
dcf4e94b-86e4-4428-8a25-15b6e60a5465	Shakespeare (Matching)	item	edit item
7461accb-800c-4b54-b187-e861f9c75720	Este Test es para la presentación de SS.II	test	edit test
05e98008-f23c-4275-b75c-21f60c7f9c0e	Shakespeare(Associate)	item	edit item

- New Test
- New Item

Ilustración 121 Elección de la pregunta a editar

Una vez elegida la pregunta nos llevará a una pantalla semejante a la de creación de la pregunta elegida, para editar los campos de una pregunta siga el tutorial de la creación de dicha pregunta ya que el funcionamiento será el mismo a diferencia de que apareceran los campos rellenos con los datos introducidos en la creación. Como ejemplo vamos a ver como sería la pantalla de edicción de la preguntas de asociación.

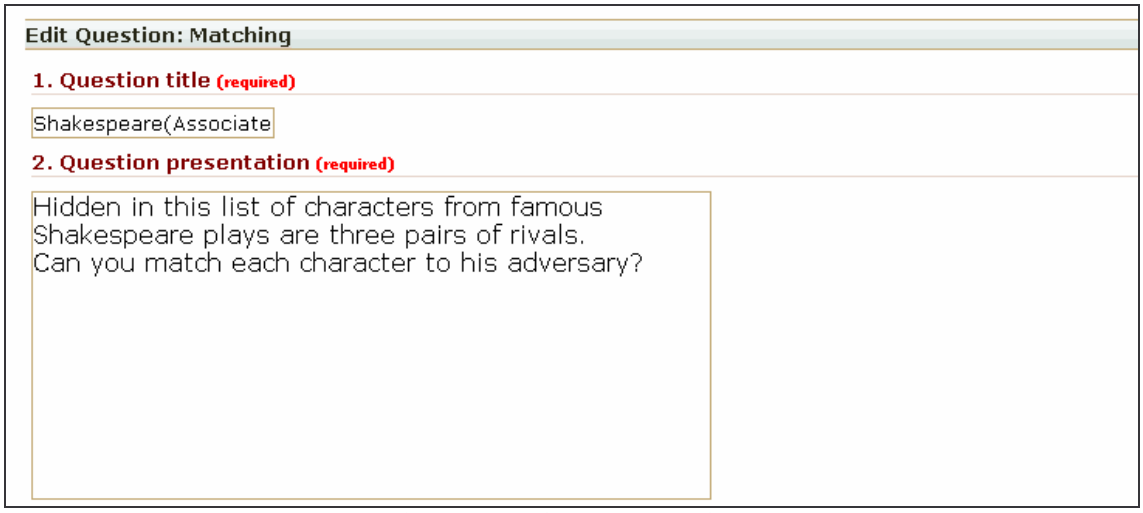


Ilustración 122 Edición de Asociación Imagen 1

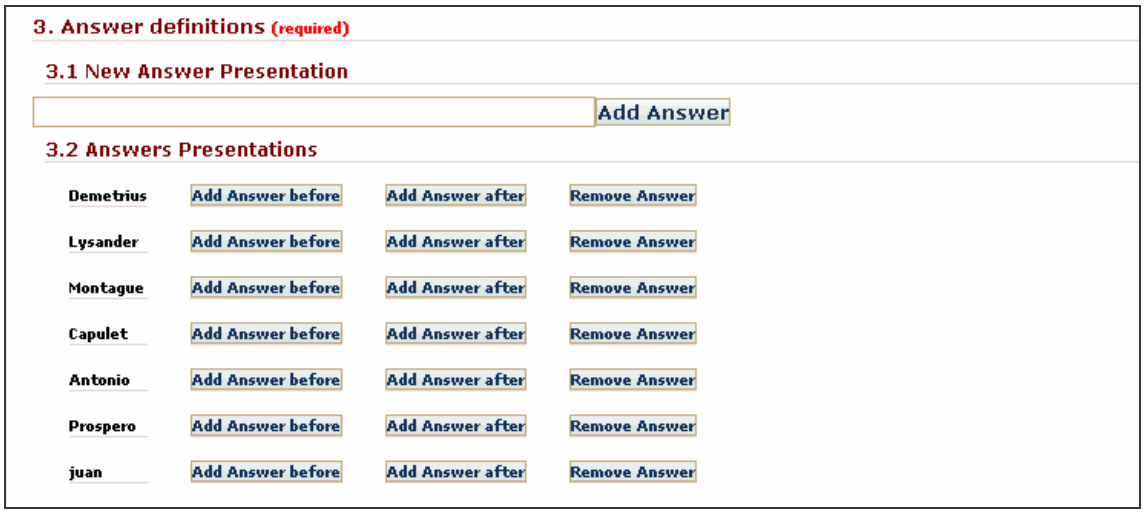


Ilustración 123 Edición de Asociación Imagen 2

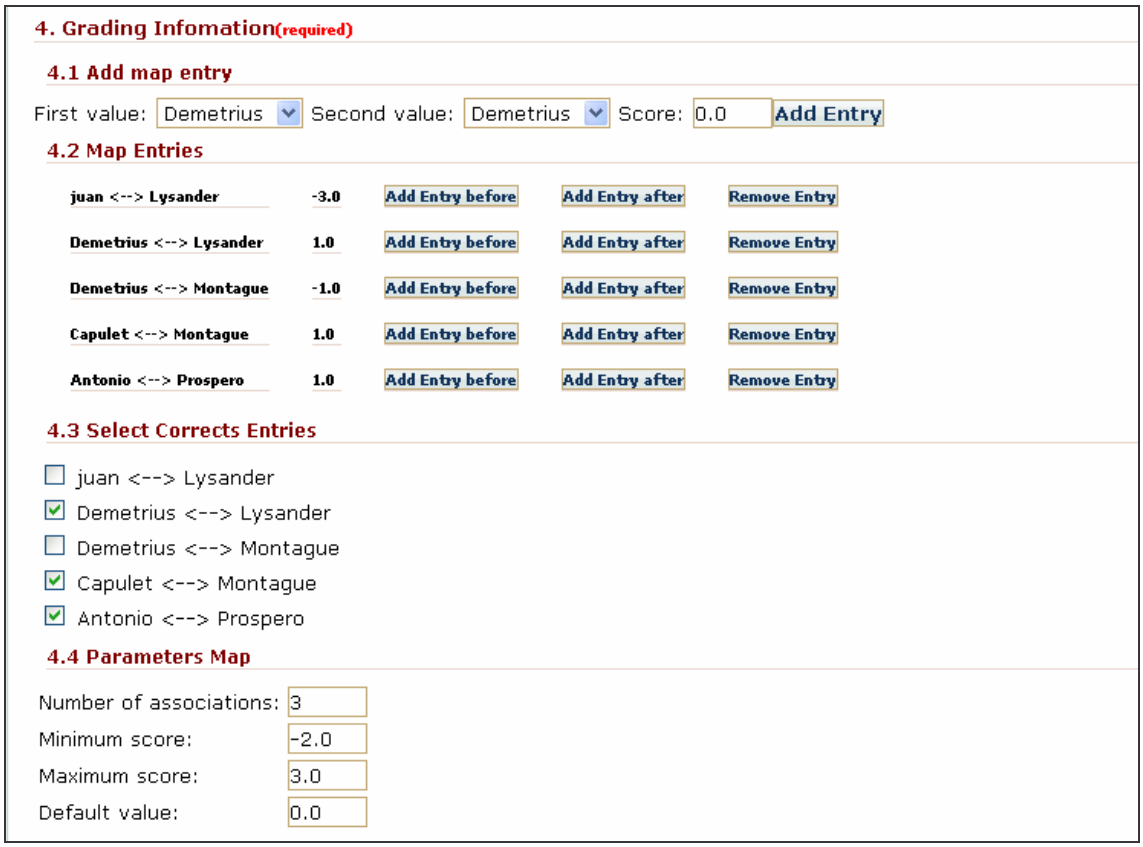


Ilustración 124 Edición de Asociación Imagen 3

Una vez editada la pregunta, para guardar los cambios presionar el botón “Update”, si se desea salir sin realizar ninguna modificación presionar el botón “Cancel”.

15.3 Manual de despliegue de la aplicación

15.3.1 ECLIPSE

1. Descargar Eclipse SDK 3.2.2. (http://www.eclipse.org/downloads/)
2. Instalación
    a. Descomprimir el fichero en C:\Archivos de programa
    b. Ejecutar Eclipse y elegir una carpeta de trabajo (workspace recomendado: C:\Documents and Settings\XXX\workspace)

15.3.2 PLUGINS para Eclipse

15.3.2.1 Subclipse 1.0.5

1. Descargar el .zip (http://subclipse.tigris.org/download.html)
2. Instalación (http://www.humbertocervantes.net/cursos/ingsoft/subclipse/subclipse.html)
    a. Abrir Eclipse
    b. Para instalar el plugin de Subclipse, ir a la *Opciones -> Help -> Software Updates -> Find and Install...*
    c. Seleccionar *Search new features to install*
    d. Seleccionar *New Remote Site...* y crear un nuevo sitio con la información siguiente:

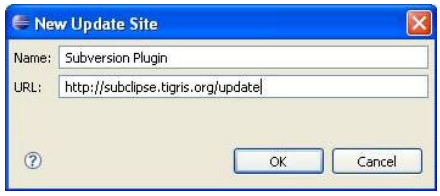


Ilustración 125. Plugins eclipse imagen 1

    e. Seleccionar el nuevo sitio y seleccionar plugin y después Finish.

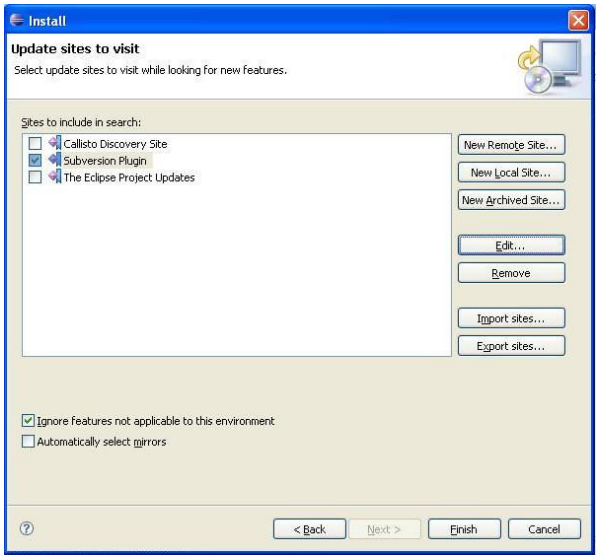


Ilustración 126 Plugins eclipse imagen 2

f. Pulsar *next*

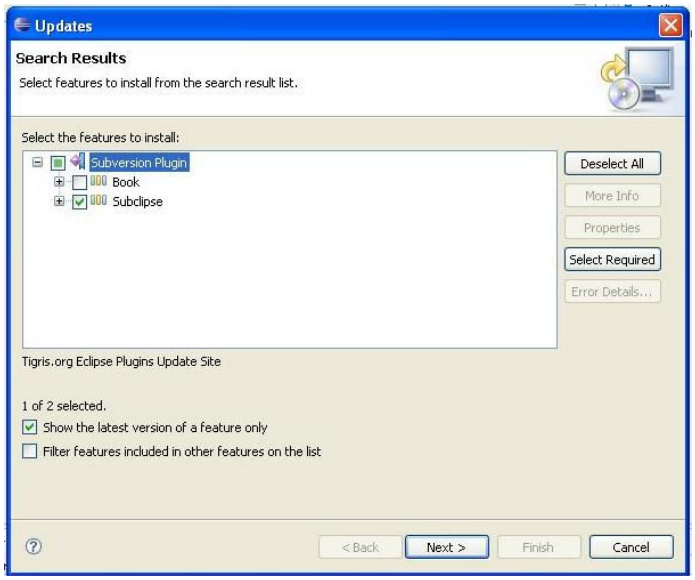


Ilustración 127 Plugins eclipse imagen 3

g. Acepar los términos de la licencia

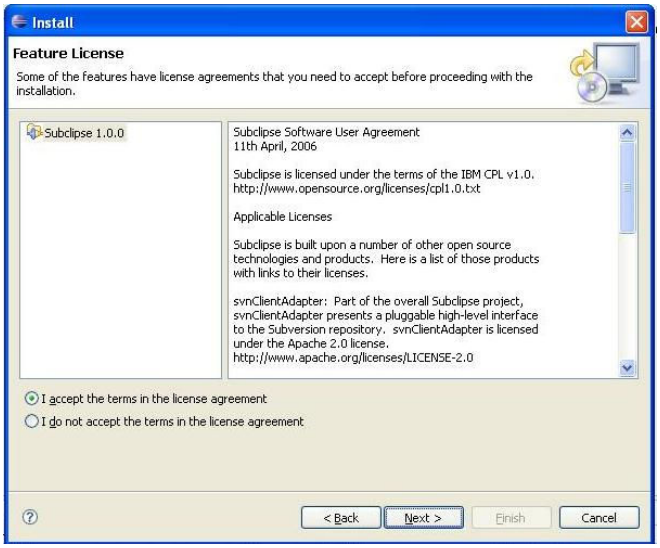


Ilustración 128 Plugins eclipse imagen 4

h. Pulsar *Next* y a continuación *Finish*

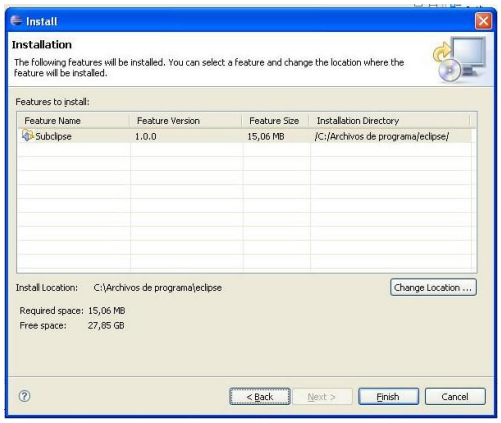


Ilustración 129 Plugins eclipse imagen 5

i. Debemos verificar que queremos instalar el Software (*Install All*)

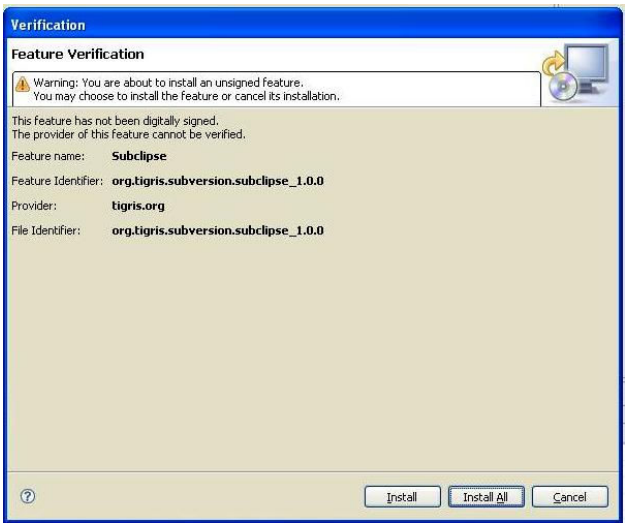


Ilustración 130 Plugins eclipse imagen 6

j. Rearrancamos el Eclipse cuando nos lo pida.

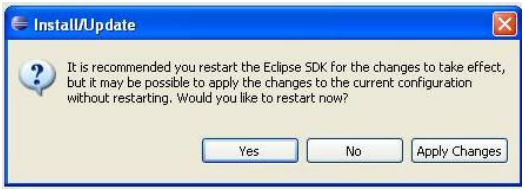


Ilustración 131 Plugins eclipse imagen 7

3. Configuración
   a. Una vez que se ha instalado el plugin, hay que configurarlo. Para ello ir a *Window -> Preferences* y seleccionar en la parte izquierda *Team -> SVN*.
   - **WINDOWS:** seleccionar *Javahl (JNI)*
   - **LINUX:** seleccionar *SVN command Line*
   Pulsar *OK*

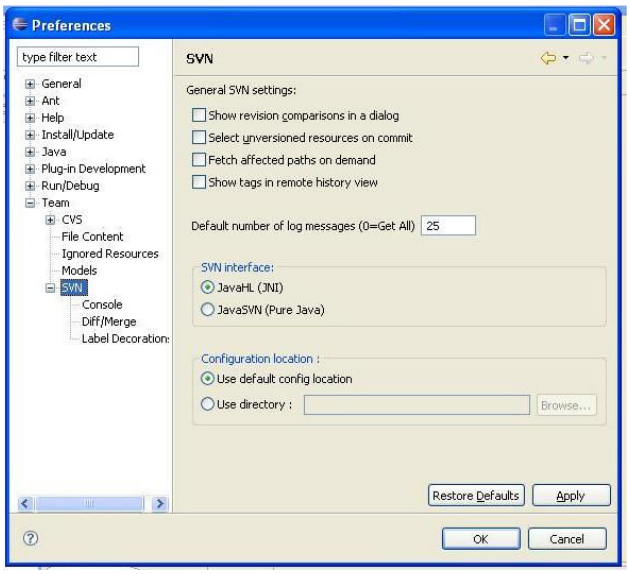


Ilustración 132 Plugins eclipse imagen 8

b. Ahora vamos a mostrar la vista de exploración del repositorio. Ir a *Window -> Show View -> Other...* y seleccionar *SVN Repository*.
c. Debe aparecer una nueva lengueta que dice *SVN Repository*, abrirla y seleccionar *SVN Repository*

Ilustración 133 Plugins eclipse imagen 9

d. En la pestaña abierta de *SVN Repository* pulsar el icono de *Add SVN Repository*

Ilustración 134 Plugins eclipse imagen 10

e. Poner los siguientes datos en la ventana que se muestra, y pulsar *Finish*.

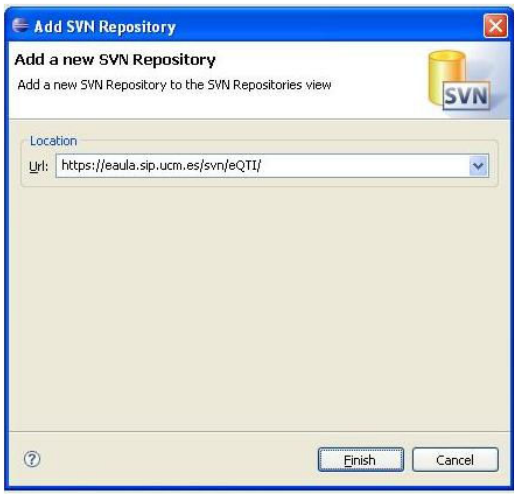


Ilustración 135 Plugins eclipse imagen 11

f. Aceptar el certificado digital pulsando sobre *Accept Permanently*

Ilustración 136 Plugins eclipse imagen 12

g. Introducir *usuario* y *password* facilitados.

Ilustración 137 Plugins eclipse imagen 13

h. Se nos mostrará un mensaje de error que debemos aceptar pulsando *Yes*.

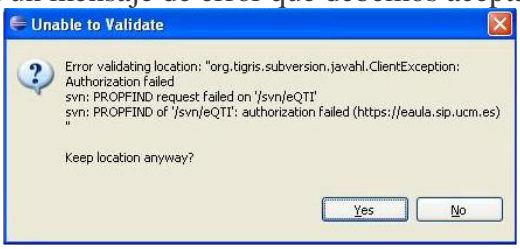


Ilustración 138 Plugins eclipse imagen 14

4. Checkout del proyecto
   a. Botón derecho en la dirección de la pestaña *SVN Repository*.

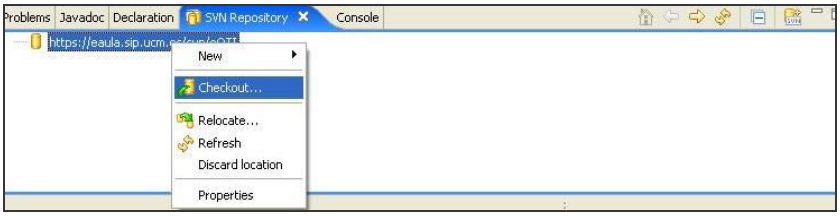


Ilustración 139 Plugins eclipse imagen 15

b. Llamamos *eQTI2* al proyecto y pulsamos *Next*

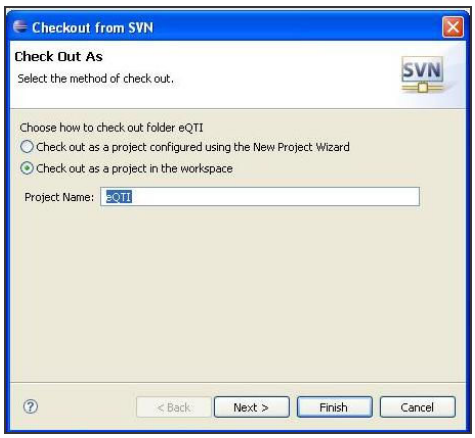


Ilustración 140 Plugins eclipse imagen 16

c. Por último, aceptamos pulsando *Finish.*

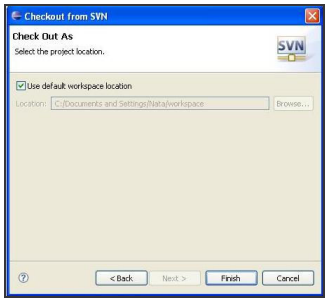


Ilustración 141 Plugins eclipse imagen 17

d. Aceptamos la descarga pulsando *Yes.*

Ilustración 142 Plugins eclipse imagen 18

e. Esperamos a que finalice la descarga (este proceso puede tardar varios minutos).

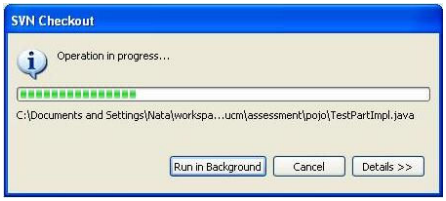


Ilustración 143 Plugins eclipse imagen 19

f. Obtenemos la siguiente vista.

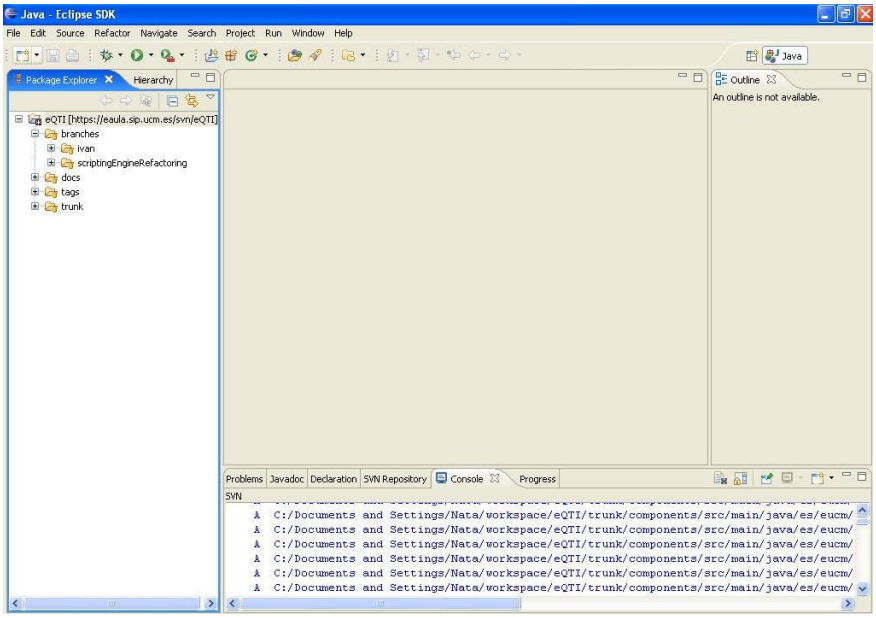


Ilustración 144 Plugins eclipse imagen 20

5. Configuración:
    a. Botón derecho sobre la raíz del proyecto -> *Properties*
    b. Desde la pestaña *Info* configuramos la siguiente información:

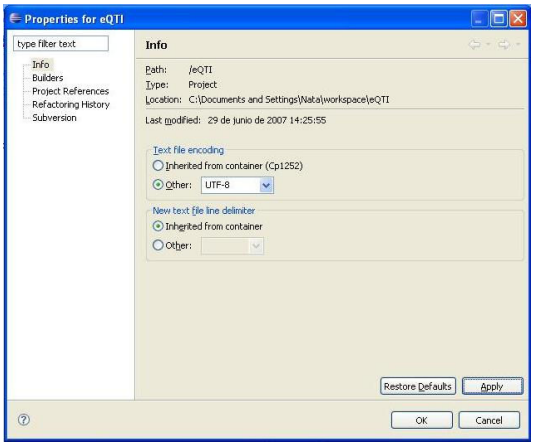


Ilustración 145 Plugins eclipse imagen 21

15.3.2.2 Tomcat Plugin

1. Descargar el archivo *tomcatPluginV32beta3.zip* de http://www.eclipsetotale.com/tomcatPlugin.html
2. Descomprimir y guardar la carpeta *com.sysdeo.eclipse.tomcat_3.2.0.beta3* en la carpeta *plugins* del directorio de Eclipse.
3. Reiniciar Eclipse.
4. Configuración:
    a. Desde Eclipse: *Windows -> Preferences -> Tomcat.*
    b. Versión 5.x y directorio del Tomcat como se indica en la figura:

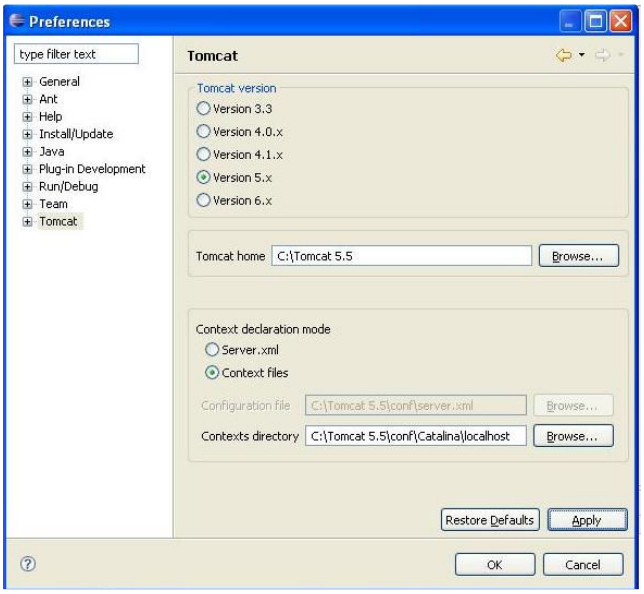


Ilustración 146 Plugins eclipse imagen 22

15.3.3 Maven

1. Descargar las extensiones para Eclipse desde http://m2eclipse.codehaus.org/
2. Crear una carpeta en el directorio de Eclipse y llamarla *eclipse_extensions.*
3. Guardar las extensiones descargadas en *eclipse_extensions/maven2Integration.*
4. Proceder de la misma manera que con el *plugin de SVN* introduciendo los siguientes datos:

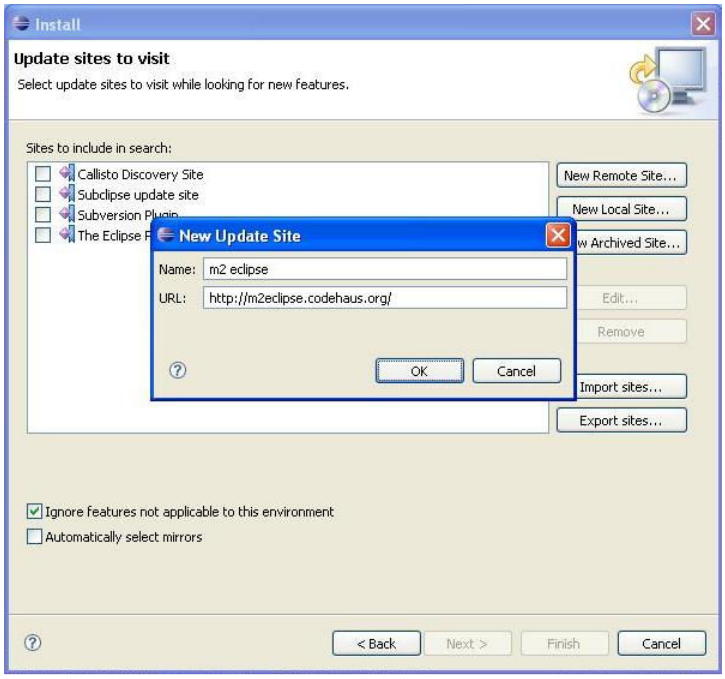


Ilustración 147 Maven imagen 1

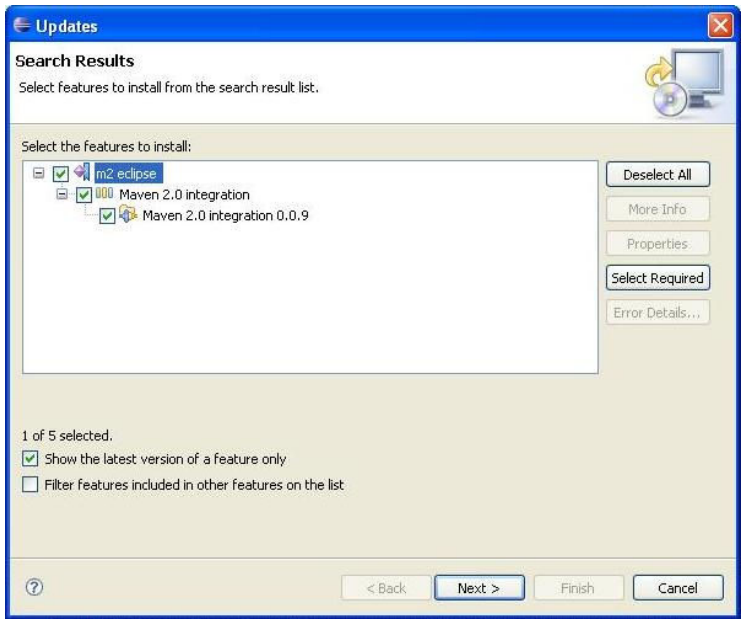


Ilustración 148 Maven imagen 2

5. Reiniciar Eclipse.
6. Descargar Maven de http://maven.apache.org/ en *C:\Archivos de programa*
7. Ejecutar *m2.bat* de *C:\Archivos de programa\maven-2.0.7\bin*
8. Boton derecho sobre el proyecto y pulsar sobre *Maven 2 enable* (este proceso puede durar unos minutos).
9. Introducir los siguientes datos y pulsar *OK.*

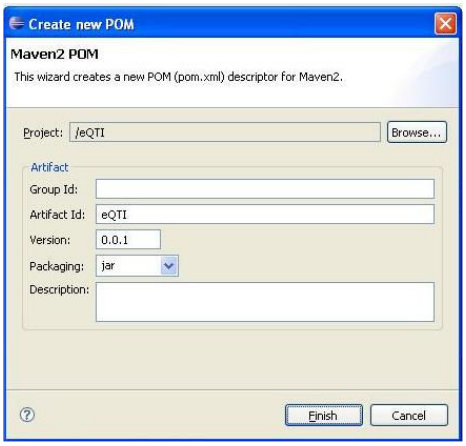


Ilustración 149 Maven imagen 3

15.3.4 MySQL

15.3.4.1 MySQL Server 5.0

1. Descargar *MySQL Server 5.0* de http://dev.mysql.com/downloads/mysql/5.0.html
2. Ejecutar el programa de instación.
3. Entrar como *root* desde *MySQL Administrator.*

Ilustración 150 MySQL imagen 1

4. Accedemos a una pantalla desde la cual creamos un nuevo usuario (user name: *eqti*, password: *eqti*).

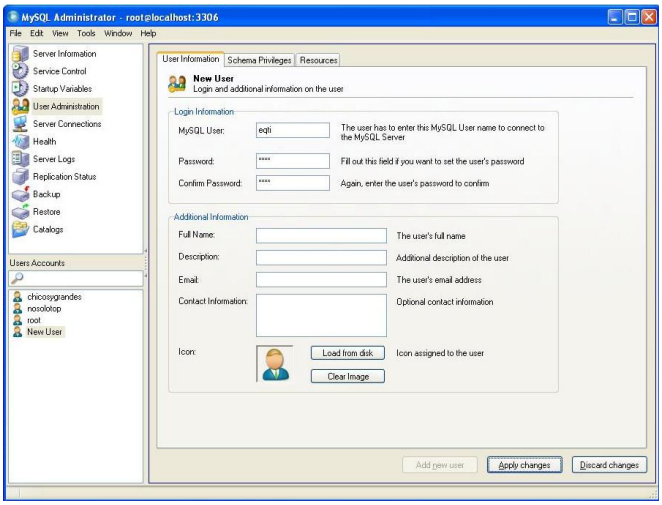


Ilustración 151 MySQL imagen 2

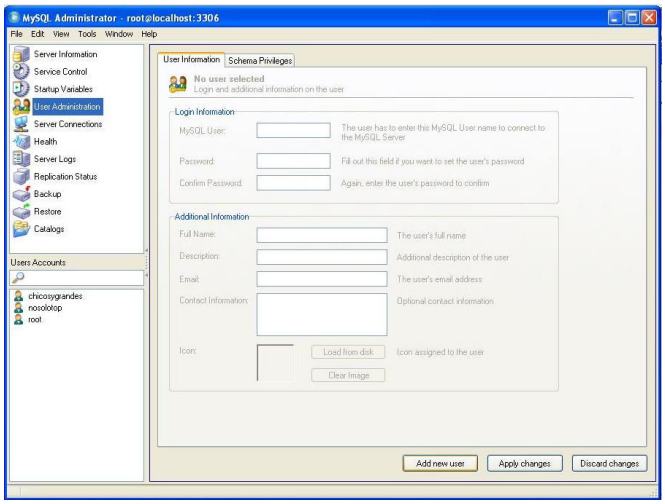
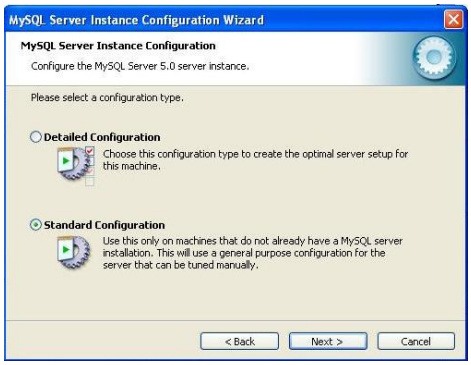


Ilustración 152 MySQL imagen 3

5. Configuración:



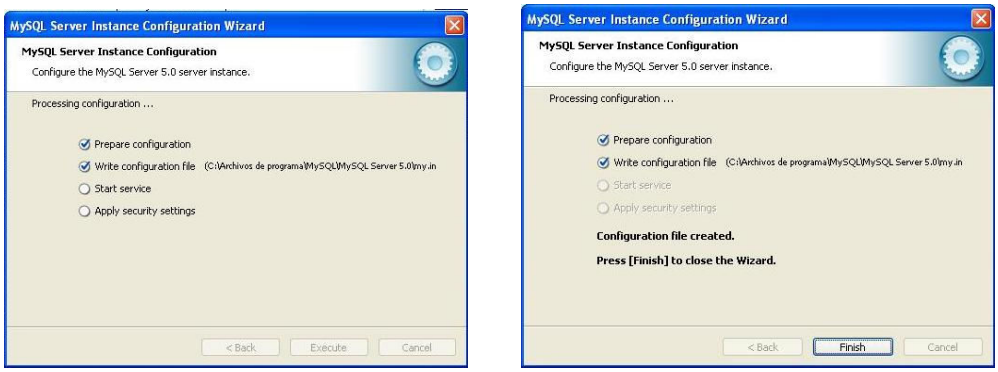


Ilustración 153 MySQL imagen 4

15.3.4.2 MySQL GUI Tools

1. Descargar *MySQL GUI Tools* de http://dev.mysql.com/downloads/mysql/5.0.html
2. Realizar la instalación completa.
3. Configuración:
    a. Abrir *MySQL Administrador.*
    b. Pulsar el botón con los tres puntos […].

Ilustración 154 MySQL imagen 5

    c. Pulsar *Connections -> New connection*.
    d. Introducir los siguientes datos:
    - Connection: *eqti*
    - Username: *eqti*
    - Password: *eqti*
    - Port: *3306*
    - Schema: *eqtiDB*

    Y pulsar *Apply*

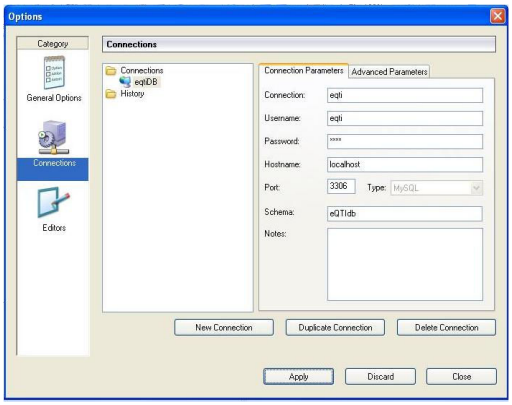


Ilustración 155 MySQL imagen 6

e. Acceder desde el panel de entrada de *MySQL administrator* con los datos del usuario y conexión creados.

Ilustración 156 MySQL imagen 7

15.3.5 Spring Framework

1. Descargarlo desde http://www.springframework.org/download y guardarlo en *C:\Archivos de programa*
2. Descomprimir el fichero *spring-src.zip* de *C:\Archivos de programa\spring-framework-2.0.1\dist* en esta mima ubicación.

15.3.5.1 Configuración

- Cambiar la expresión *CHANGE THIS!!!* por *C:/Documents and Settings/XXX/.m2/repository* en el fichero *build.xml* ubicado en *C:\Documents and Settings\Nata\workspace\eQTI2\components*
- Copiar los ficheros *hibernate.properties.sample* y *build.properties.sample* dejándolos con extensión *.properties*
- Configurar el fichero *build.properties* para que apunte a *C:/Documents and Settings/XXX/.m2/repository*
- Crear un fichero *settings.xml* en *C:\Documents and Settings\Nata\.m2* con el siguiente contenido:

```
<settings>
  <mirrors>
    <mirror>
        <id>internal-repository</id>
          <name>UCM Internal Repository</name>
            <url>http://eaula.sip.ucm.es/mavenproxy/repository</url>
              <mirrorOf>central</mirrorOf>
    </mirror>
    <mirror>
        <id>internal-repository2</id>
          <name>UCM Internal Repository</name>
            <url>http://eaula.sip.ucm.es/maven-proxy/repository</url>
              <mirrorOf>dev-java-net</mirrorOf>
   </mirror>
  </mirrors>
</settings>
```

- Crear un fichero *eqti-users.xml* en *C:\tomcat\conf* con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
  <tomcat-users>
    <role rolename="ADMIN"/>
    <role rolename="TUTOR"/>
    <role rolename="LEARNER"/>
    <user username="admin" password="admin" roles="ADMIN,TUTOR,LEANER"/>
    <user username="tutor" password="tutor" roles="TUTOR,LEANER"/>
    <user username="learner" password="learner" roles="LEARNER"/>
  </tomcat-users>
```

- Crear un fichero *eQTI.xml* en *C:\tomcat\conf\Catalina\localhost* con el siguiente contenido:

```
<Context path="/eQTI" reloadable="true"
       docBase="C:\Documents and
Settings\XXX\workspace\eQTI2\tool\target\eQTI-Tool-1.0"
       workDir="C:\Documents and Settings\XXX\workspace\eQTI2\work"
debug="99">

  <Resource name="jdbc/eqtiDB"
            auth="Container"
            type="javax.sql.DataSource"
            maxActive="50" maxIdle="10" maxWait="10000"
```

```
            username="eqti" password="eqti"
driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/eqtiDB?autoReconnect=true"/>

  <Realm className="org.apache.catalina.realm.MemoryRealm"
         pathname="conf/eqti-users.xml"
        debug="99" />
</Context>
```

- Editar el archivo *C:\tomcat\conf\Catalina\localhost\eQTI.xml*, haciendo que el *docBase* y *work* apunten al directorio correcto donde está el proyecto de Eclipse.
- Dentro del directorio *C:\Documents and Settings\Nata\.m2\repository* borrar la carpeta *hibernate*.
- Hacer un *checkout* completo del proyecto.
  - Compilar el proyecto (si da algún fallo ir a *Project->Build Project*)
  - Arrancar MySQL
- Abrir una consola *CMD* y hacer ("…" apunta a la ruta del proyecto eQTI):
  C:\...\common>mvn install
  C:\...\api>mvn install
  C:\...\components>mvn install
  C:\...\components>ant schema-recreate load-data
  C:\...\components>mvn install
  C:\...\tool>mvn compile war:exploded
- Desde Eclipse:
  - *Project->Clean->eQTI2*
  - Seleccionar el archivo *build.xml* que está en el raíz y selecionar *Run As -> Ant Build* con el botón derecho.
- Arrancar Tomcat desde Eclipse.
- Abrir un navegador y probar http://localhost:8080/eQTI

16 Anexo A

16.1 Tutorial JSF-Library Web

El primer paso es crearnos un nuevo proyecto para ello le damos a:
File->New->Project

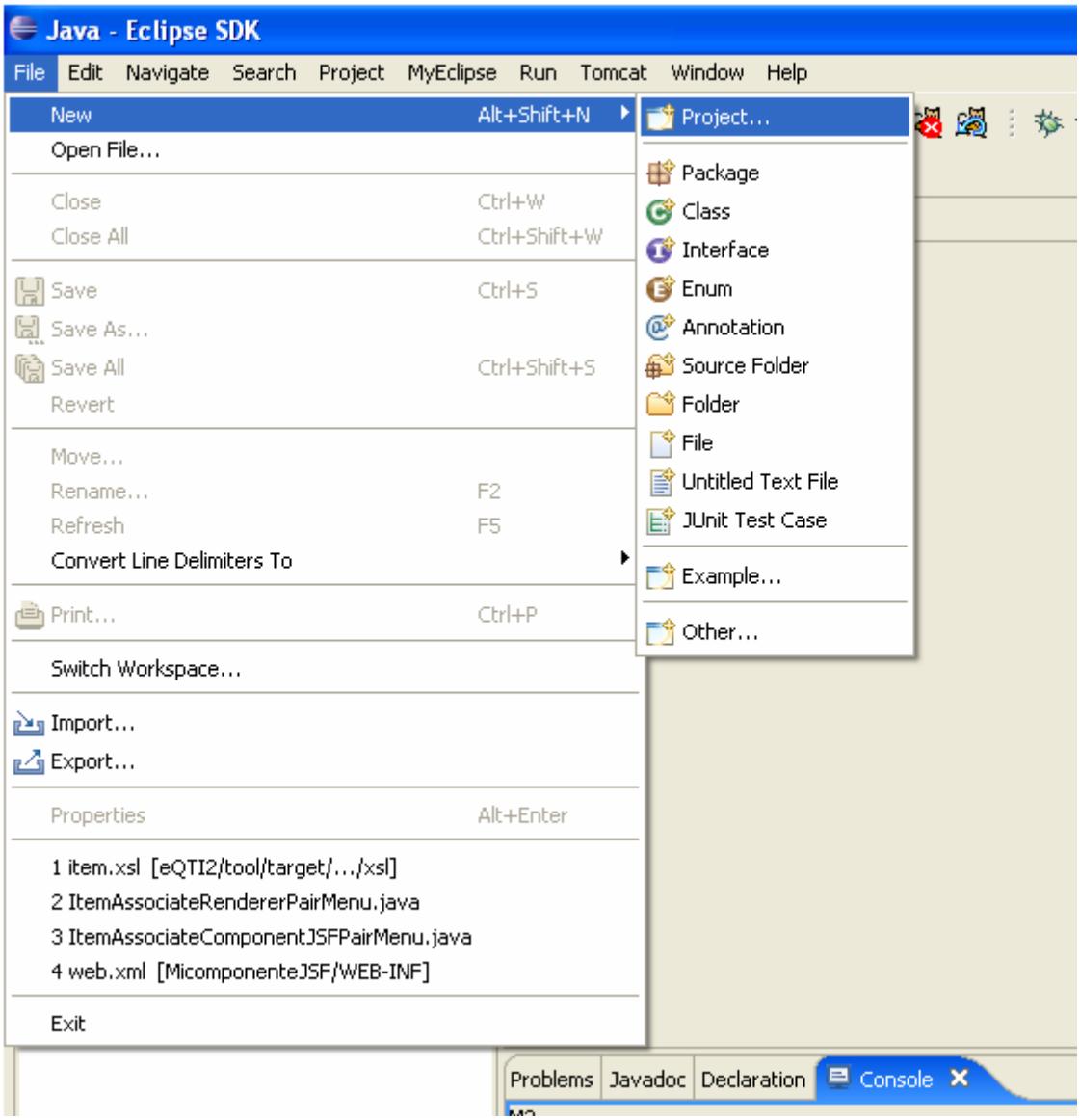


Ilustración 157: Imagen tutorial JSF 1

Elegir un proyecto Tomcat Project que se encuentra dentro de la carpeta Java. Presionar el botón Next , elegir un nombre para nuestro proyecto y después elegir el

wokspace(directorio donde se encuentra la carpeta del proyecto).Por ultimo presionar el botón Finish.

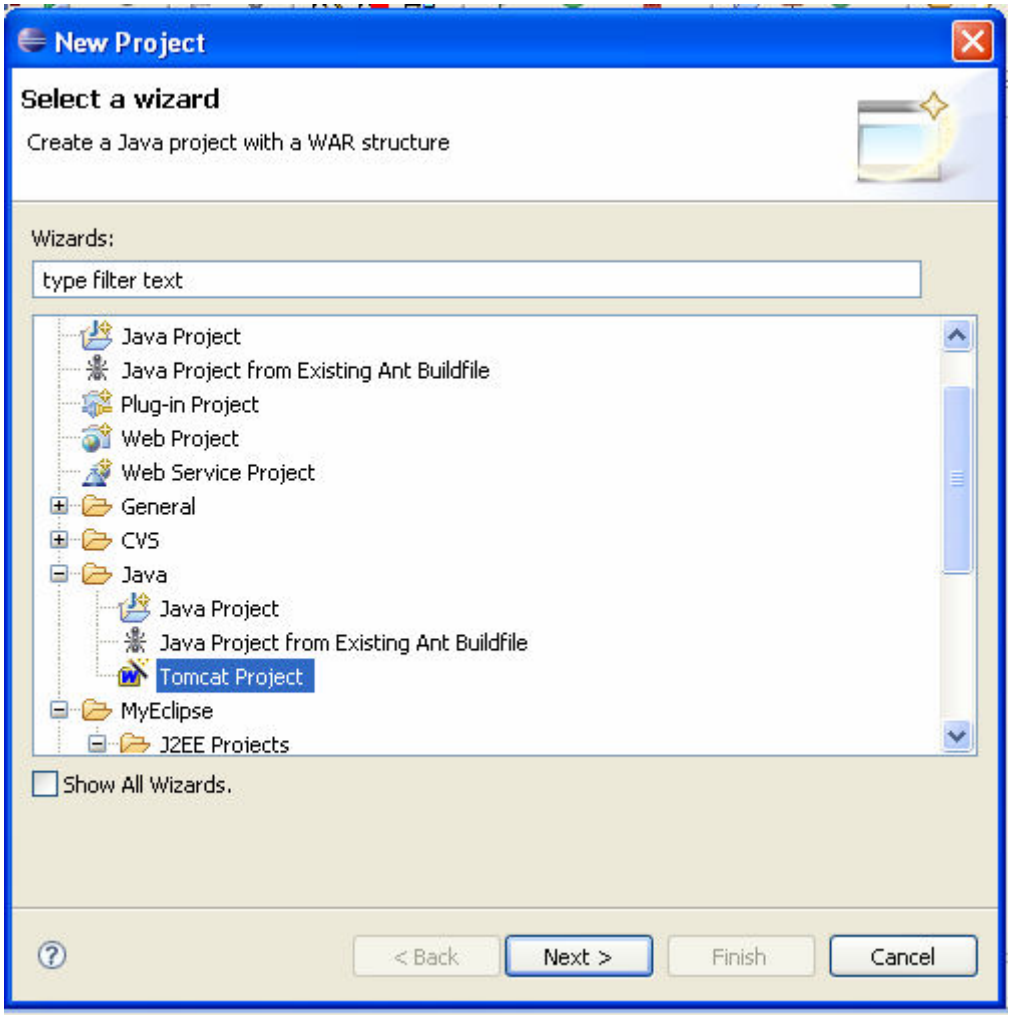


Ilustración 158: Imagen tutorial JSF 2

Si todo ha ido bien nos debe salir una estructura de archivos como esta:

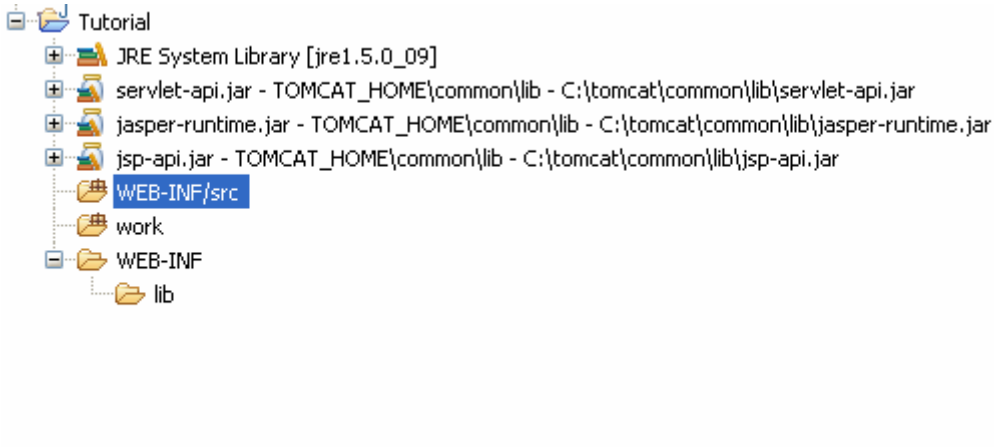


Ilustración 159: Imagen tutorial JSF 3

La carpeta WEB-INF/src es la carpeta donde debemos poner nuestros archivos .java, la carpeta WEB-INF/lib es el lugar donde deberemos colocar los .jar que necesitamos para el proyecto que estamos creando. Para ello arrastrar los .jar que necesitamos a la carpeta desde el explorador de windows y una vez que se encuentran allí, vamos a nuestro proyecto de eclipse y presionamos con el botón derecho a nuestro proyecto y le damos a refrescar, y nos debe aparecer los jar ya incluidos.

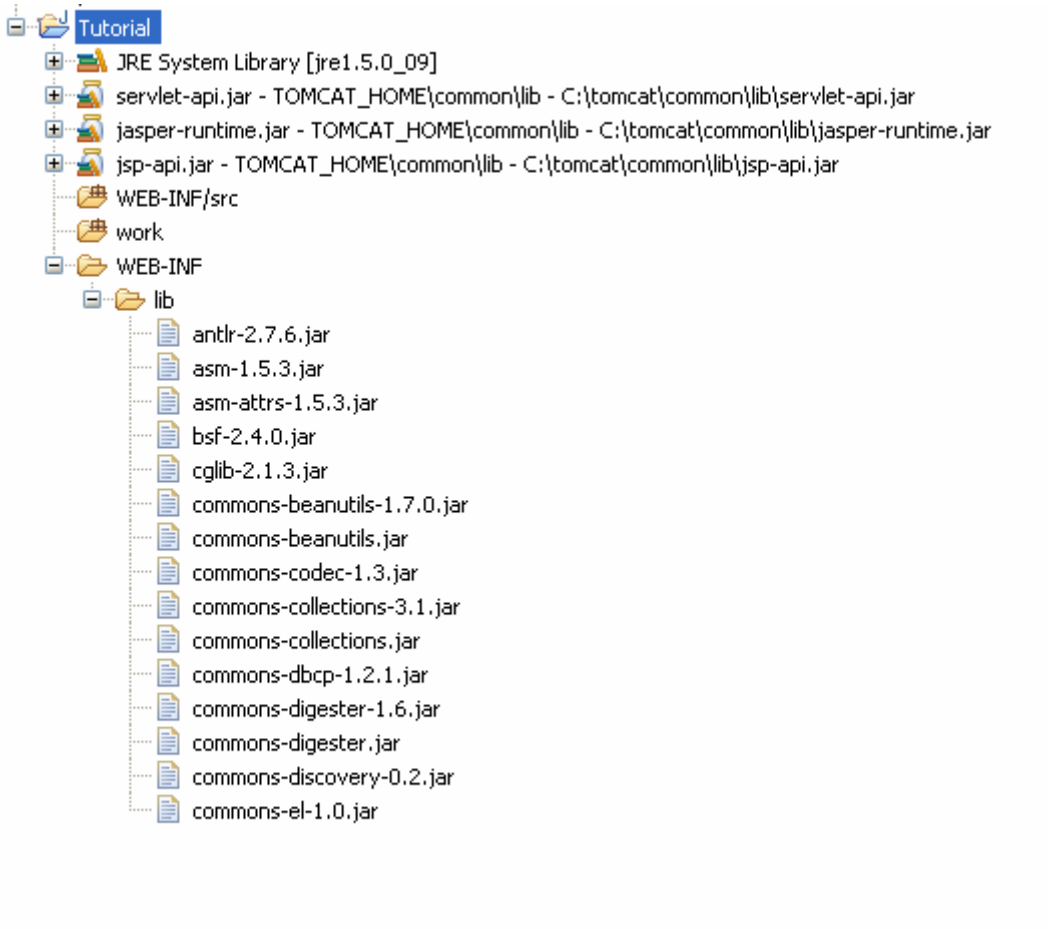


Ilustración 160: Imagen tutorial JSF 4

Por último solo nos queda crear los archivos faces-config.xml y web.xml de nuestro proyecto, para crearlos presionar con el botón derecho del ratón en la carpeta WEB-INF y le damos a: New-> File y allí ponemos el nombre de nuestro fichero con su extensión.

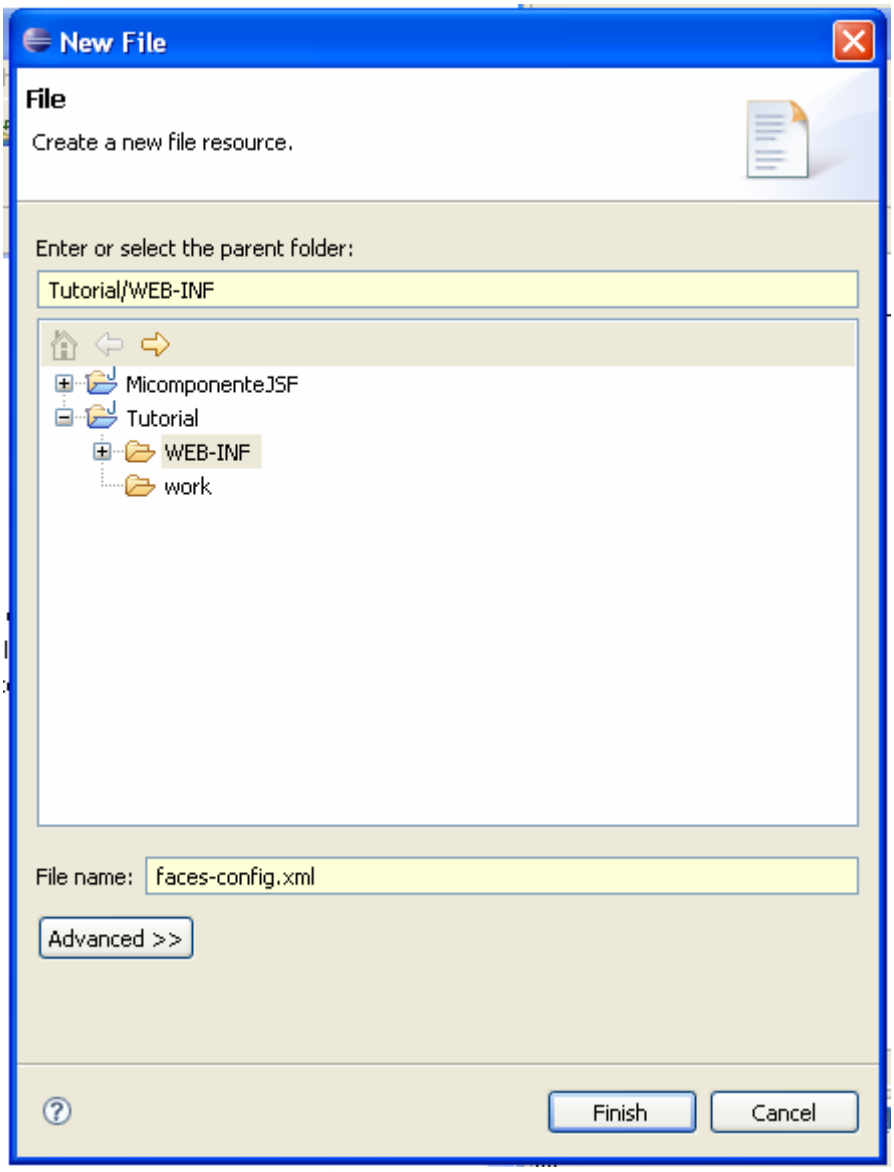


Ilustración 161: Imagen tutorial JSF 5

En la carpeta src, presionar con el botón derecho y en new->class crear una clase llamada Book.java, a continuación rellenar la clase con el siguiente código:

```
import java.io.Serializable;
import java.util.Map;
import javax.faces.component.UIParameter;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;

public class Book implements Serializable {

      private long id;
      private String author;
      private String title;
      private boolean available;

      public Book(){}
      public Book(long id, String author, String title, boolean
available){
            this.id = id;
            this.author = author;
            this.title = title;
            this.available = available;
      }

public String getAuthor() {
            return author;}

public void setAuthor(String author) {
            this.author = author;}

      public boolean isAvailable() {
            return available;}

public void setAvailable(boolean available) {
            this.available = available;}

public long getId() {
            return id;}

public void setId(long id) {
            this.id = id;     }

public String getTitle() {
            return title;     }

public void setTitle(String title) {
            this.title = title;     }


public void setBook(Book book){
            this.setId(book.getId());
            this.setAuthor(book.getAuthor());
            this.setTitle(book.getTitle());
            this.setAvailable(book.isAvailable());}

public Book getBook(){
      return new Book(this.getId(), this.getAuthor(),
            this.getTitle(),this.isAvailable());}
```

```
public void initBook(ActionEvent event){

            this.setBook(new Book());
      }

public void selectBook(ActionEvent event){

            SimulateDB simulateDB = new SimulateDB();
            Map session =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap()
;
            UIParameter component = (UIParameter)
event.getComponent().findComponent("editId");
      long id = Long.parseLong(component.getValue().toString());
      this.setBook(simulateDB.loadBookById(id, session));
      }

public void saveBook(ActionEvent event){

            SimulateDB simulateDB = new SimulateDB();


            Map session =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap()
;

            simulateDB.saveToDB(this.getBook(), session);
      }


public void deleteBook(ActionEvent event){

            SimulateDB simulateDB = new SimulateDB();


            Map session =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap()
;


            UIParameter component = (UIParameter)
event.getComponent().findComponent("deleteId");


            long id = Long.parseLong(component.getValue().toString());


            simulateDB.deleteBookById(id, session);
      }

}
```

De igual forma vamos a crear nuestra clase SimulateDB.java, esta clase lo que nos va a hacer es simularnos una base de datos donde están guardados nuestros libros, una vez creada rellenar la clase con el siguiente código:

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.Map;
import java.util.Random;


public class SimulateDB {

    private Collection books;


    private void init(Map session) {
        books = new ArrayList();
        Random random = new Random();
        books.add(new Book(Math.abs(random.nextLong()), "David
Roos", "Struts book", true));
        books.add(new Book(Math.abs(random.nextLong()), "Micheal
Jackson", "Java book", true));
        books.add(new Book(Math.abs(random.nextLong()), "Bruce
Lee", "Java2 book", false));
        books.add(new Book(Math.abs(random.nextLong()), "Tom
Jones" ,"EJB book", true));
        books.add(new Book(Math.abs(random.nextLong()), "Mc
Donald", "Jboss for beginners", false));
        books.add(new Book(Math.abs(random.nextLong()), "Lars
Mars", "Using Myeclipse for cooking", true));
        books.add(new Book(Math.abs(random.nextLong()), "Mary
Jane", "EJB or spending your weekends", true));

        session.put("bookDB", books);
    }

private void loadData(Map session) {
        books = (Collection) session.get("bookDB");
        if (books == null)
            init(session);
}

private void saveData(Map session) {
        session.put("bookDB", books);}
            Random random = new Random();
            book.setId(random.nextLong());
            books.add(book);
        }
```

```
public long saveToDB(Book book, Map session) {

        loadData(session);

        boolean bookExist = false;
        ArrayList booksNew = (ArrayList) books;
        int index = 0;
        for (Iterator iter = books.iterator(); iter.hasNext();) {
            Book element = (Book) iter.next();
            if (element.getId() == book.getId()) {
                  booksNew.set(index, book);
                bookExist = true;
                break;
            }
            index++;
        }

        books = booksNew;
        if (bookExist == false) {
            Random random = new Random();
            book.setId(random.nextLong());
            books.add(book);
        }
        saveData(session);

        return book.getId();
    }

public Book loadBookById(long id, Map session) {
        loadData(session);
        for (Iterator iter = books.iterator(); iter.hasNext();) {
            Book element = (Book) iter.next();
            if (element.getId() == id) return (Book) element;
        }
        return null;
    }

public void deleteBookById(long id, Map session){

        loadData(session);
        Collection booksNew = new ArrayList();
        for (Iterator iter = books.iterator(); iter.hasNext();) {
            Book element = (Book) iter.next();
            if (element.getId() != id){
                  booksNew.add(element);
            }
        }
        books = booksNew;
        saveData(session);
    }

    public Collection getAllBooks(Map session) {
        loadData(session);
        return books;

    }
}
```

También crear de igual forma la clase BookList.java con el siguiente código:

```
import java.util.Collection;
import java.util.Map;

import javax.faces.context.FacesContext;

public class BookList {
      Collection books;
      public Collection getBooks(){

            SimulateDB simulateDB = new SimulateDB();


            Map session =
FacesContext.getCurrentInstance().getExternalContext().getSessionMap()
;
            books = simulateDB.getAllBooks(session);

            return books;
      }

public void setBooks(Collection books) {
            this.books = books;
      }
}
```

Una vez creadas todas nuestras clases vamos a pasar a crear nuestras paginas jsp del ejemplo, las paginas jsp deben estar a la misma altura que al carpeta WEB-INF y no dentro para ello presionamos con el botón derecho en nuestro proyecto y le damos a new->File y creamos nuestra primera página que va a ser index.jsp, una vez creada le introducimos el siguiente código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
  <body>
    <jsp:forward page="/listBooks.faces" />
  </body>
</html>
```

A continuación crearemos la página listbook.jsp con el siguiente código:

```
<%@ page language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getS
erverPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <base href="<%=basePath%>">

      <title>List of books</title>
</head>

<body>
      <f:view>
            <h:form id="bookList">
            <h:dataTable id="books"
                              value="#{bookListBean.books}"
                              var="book"
                              border="1">
              <h:column>
                <f:facet name="header">
                 <h:outputText  value="Author"/>
                </f:facet>
                <h:outputText value="#{book.author}" />
              </h:column>
              <h:column>
                <f:facet name="header">
                  <h:outputText  value="Title"/>
                </f:facet>
                <h:outputText value="#{book.title}" />
              </h:column>
              <h:column>
                <f:facet name="header">
                  <h:outputText  value="Available"/>
                </f:facet>
                <h:selectBooleanCheckbox disabled="true"

value="#{book.available}" />
              </h:column>
```

```
<h:column>
                <f:facet name="header">
                  <h:outputText  value="Edit"/>
                </f:facet>
                <h:commandLink id="Edit"
                                action="editBook"

actionListener="#{bookBean.selectBook}">
                  <h:outputText value="Edit" />
                  <f:param id="editId"
                              name="id"
                              value="#{book.id}" />
                </h:commandLink>
              </h:column>
               <h:column>
                <f:facet name="header">
                  <h:outputText  value="Delete"/>
                </f:facet>
                <h:commandLink id="Delete"
                                action="listBooks"

actionListener="#{bookBean.deleteBook}">
                  <h:outputText value="Delete" />
                  <f:param id="deleteId"
                              name="id"
                              value="#{book.id}" />
                </h:commandLink>
              </h:column>
            </h:dataTable>

            <h:commandLink id="Add"
                              action="editBook"

actionListener="#{bookBean.initBook}">
                  <h:outputText value="Add a book" />
            </h:commandLink>
            </h:form>
      </f:view>
</body>
</html>
```

Y por último crearemos nuestra página editBook.jsp con el siguiente código:

```
<%@ page language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getSe
rverPort()+path+"/";
%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <base href="<%=basePath%>">

      <title>Add / Edit a book</title>
</head>

<body>
      <f:view>
            <h:form>
            <h:inputHidden id="id" value="#{bookBean.id}"/>
            <h:panelGrid columns="2" border="1">

                  <h:outputText value="Author:" />
                  <h:inputText id="author"
                                    value="#{bookBean.author}">
                  </h:inputText>

                <h:outputText value="Title:" />
                  <h:inputText id="title"
                                   value="#{bookBean.title}">
                  </h:inputText>

                  <h:outputText value="Available:" />
                  <h:selectBooleanCheckbox id="available"

value="#{bookBean.available}" />

            </h:panelGrid>

            <h:messages id="errors"
                              style="color:red;font-weight:bold"
                              layout="table" />

            <h:commandButton value="Save"
                                    action="listBooks"

actionListener="#{bookBean.saveBook}" />
            </h:form>
      </f:view>
</body>
</html>
```

Como último paso ya sólo nos queda editar nuestros archivos faces-config.xml y web.xml, el primero debe tener un código similar a este, tendiendo en cuenta si habéis decidido meter las clase java en algún paquete adicional.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD
JavaServer Faces Config 1.1//EN" "http://java.sun.com/dtd/web-
facesconfig_1_1.dtd">
<faces-config>

      <!-- Navigation rules -->
      <navigation-rule>
            <description>List of books</description>
            <from-view-id>/listBooks.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>editBook</from-outcome>
                  <to-view-id>/editBook.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>

      <navigation-rule>
            <description>Add or edit a book</description>
            <from-view-id>/editBook.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>listBooks</from-outcome>
                  <to-view-id>/listBooks.jsp</to-view-id>
                  <redirect/>
            </navigation-case>
      </navigation-rule>

      <!-- Managed beans -->
      <managed-bean>
            <description>
                  Book bean
            </description>
            <managed-bean-name>bookBean</managed-bean-name>
            <managed-bean-class>Book</managed-bean-class>
            <managed-bean-scope>request</managed-bean-scope>
      </managed-bean>

      <managed-bean>
            <description>
                  BookList Bean
            </description>
            <managed-bean-name>bookListBean</managed-bean-name>
            <managed-bean-class>BookList</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>

</faces-config>
```

Y nuestro archivo web.xml debe ser de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="2.4" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>

            <listener>
            <listener-class>

      org.apache.myfaces.webapp.StartupServletContextListener
            </listener-class>
            </listener>


  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
</web-app>
```

Para probarla solo nos queda presionar con el botón derecho en el proyecto y darle a: Run->Run As y seleccionar nuestro proyecto, a continuación arrancar desde nuestro eclipse el Tomcat presionando su botón

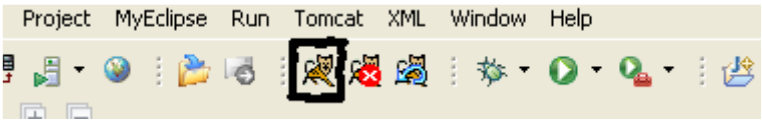


Ilustración 162: Imagen tutorial JSF 7

y por último abrir nuestro navegador y poner la siguiente url:

http://localhost:8080/nombreDeMiProyecto

Donde nombreDeMiProyecto es el nombre que le hemos dado al proyecto creado con este tutorial.

16.2 Spring-Hibernate

HolaMundo

Consiste en un ejemplo muy sencillo que muestra como a través del archivo de configuración de Spring podemos definir qué clase implementa cada interfaz y con qué configuración inicial.

Spring actúa como una factoría que nos devuelve el bean que le indiquemos de la siguiente forma:

- Primero: Se crea la factoría que se encarga de cargar todos los beans especificados en el archivo de configuración.

  ClassPathXmlApplicationContext factory = new ClassPathXmlApplicationContext("Hola.xml");

  ClassPathXmlApplicationContext carga las definiciones del contexto desde un archivo del class path .

- Segundo: Invocamos a la factoría con el método "getBean" pasándole como argumento el nombre del bean que deseemos crear (no nos debemos olvidar de hacer el casting).

  HolaMundo holaMundo = (HolaMundo) factory.getBean("HolaMundo");

La configuración se cargará mediante un archivo xml en el que se definirán todas las relaciones.

//Cabecera en la que se indica el tipo de archivo y de dónde se cargan las dtd de los beans.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
```

//Definición de los beans

```
<beans>
      <bean id="HolaMundo" class="hola.HolaImpl1"></bean>
</beans>
```

id : Identificador del bean
class: Clase que implementa el bean indicado.

Por defecto todos los bean de Spring siguen el patrón Singleton, en el caso de que queramos una instancia por cada bean que solicitemos debemos establecer su propiedad singleton a false.

```
<beans>
      <bean id="HolaMundo" class="hola.HolaImpl1" singleton= "false">
      </bean>
</beans>
```

Esta forma de definir los beans nos permite que modificando simplemente el archivo de configuración cambiemos las clases que los implementan y las relaciones entre ellos como explicare más adelante.

En cada bean también podemos definir qué métodos queremos que se ejecuten al crearlos (init-method) o al destruirlos (destroy-method) en sus propiedades:

```
<bean id="miBean" class="MiBeanImpl"
      init-method="nombreMetodoAlCrear"
      destroy-method="nombreMetodoAlDestruir"/>
```

Podemos inyectar las dependencias mediante los métodos setter, ésta será la forma que se utilizará en el proyecto:

```
<beans>
      <bean id="HolaMundo" class="hola.HolaImpl1">
            <property name="mensaje">
                  <value>Hola Mundo!</value>
            </property>
      </bean>
</beans>
```

En el ejemplo se establece la propiedad (atributo) mensaje con el valor indicado, para ello realiza una llamada a la función set correspondiente (setPropiedad(valor)), en éste caso setMensaje(“Hola Mundo!”). En esta relación también se puede hacer referencia a otro bean a través de la etiqueta <ref> dentro de la propiedad.

```
<bean id="bean1" class="Bean1Impl">
      <property name="propiedad1">
            <ref bean = bean2>
      </property>
</bean>
<bean id="bean2" class="Bean2Impl"></bean>
```

Como valores de una propiedad también podemos pasar alguna de las siguientes colecciones típicas en Java:

XML	Type
<list>	java.awt.List, arrays
<set>	java.awt.Set
<map>	java.awt.Map
<props>	java.awt.Properties

Ilustración 163: Imagen tutorial Spring-Hibernate 1

<List>

```
<bean id="HolaMundo" class="hola.HolaImpl2">
      <property name="mensaje">
            <list>
                  <value>Hola</value>
                  <value>Mundo</value>
                  <value>2!!!</value>
            </list>
      </property>
</bean>
```

<Set>

```
<bean id="Bean" class="BeanImpl">
      <property mane="propiedadSet">
            <set>
                  <value>valor1</value>
                  <ref bean="otroBean"/>
            </set>
      </property>
</bean>
```

<Map>

```
<bean id="Bean" class="BeanImpl">
      <property mane="propiedadMap">
            <map>
                  <entry key="key1">
                        <value>valor1</value>
                  </entry>
                  <entry key="key2">
                        <ref bean="otroBean"/>
                  </entry>
            </map>
      </property>
</bean>
```

<Props>

```
<bean id="Bean" class="BeanImpl">
      <property mane="propiedadProps">
            <props>
                  <props key="key1">valor1</props>
                  <props key="key2">valor2</props>
            </props>
      </property>
</bean>
```

Si lo que queremos es introducir un valor null simplemente introducimos como valor el elemento `<null/>`

Otra forma de la que podemos definir las dependencias entre los beans es a través de las constructoras indicándoles los parámetros.

```
<bean id="bean1" class="Bean1Impl">
      <constructor-arg>
            <value>13</value>
      </constructor-arg>
</bean>

<bean id="bean2" class="Bean2Impl">
      <constructor-arg>
            <ref bean = bean1>
      </constructor-arg>
</bean>
```

Spring seleccionará el constructor que pueda encajar (pudiendo realizar algún casting) con los parámetros introducidos sin tener en cuenta el orden de los mismos, a no ser que se indexe cada parámetro o se indique el tipo exacto del mismo.

```
<bean id="bean3" class="Bean3Impl">
      <constructor-arg index="1">
            <value>valor1</value>
      </constructor-arg>
      <constructor-arg index="0">
            <value>valor0</value>
      </constructor-arg>
</bean>

<bean id="bean4" class="Bean4Impl">
      <constructor-arg type="tipo1">
            <value>valor1</value>
      </constructor-arg>
      <constructor-arg type="tipo2">
            <value>valor2</value>
      </constructor-arg>
</bean>
```

El ejemplo de HolaMundo cuenta con un archivo de configuración (Hola.xml) en el que se indican diferentes tipos de configuración para un objeto del tipo “HolaMundo”, éste puede ser implementado por diferentes clases: “HolaImpl1” que utiliza un mensaje de tipo String y “HolaImpl2” que utiliza un mensaje de tipo List. En el caso de “HolaImpl2” se puede indicar qué clase se desea que implemente el mensaje (del tipo List) mediante el bean “Lista”.

Nota: Para probar las diferentes configuraciones basta con comentar el bean actual y descomentar la configuración deseada en “Hola.xml”.
Para que el proyecto eclipse funcione se deben añadir las librerías de la carpeta “lib” al path.

Registro de Usuarios

En este ejemplo ilustramos un registro simple de un usuario en un sistema en el que sólo debe introducir su nombre, por cada nombre registrado se devolverá un número entero que será el identificador del usuario.

En el proyecto se definen dos interfaces: “dao.UsuarioDao” que describe los servicios de la base de datos (en nuestro ejemplo será una base de datos estática implementada por “dao.impl1.UsuarioDaoStatic” y por “dao.impl2.UsuarioDaoStatic2”) y la interfaz “registrar.InterfazRegistro” que describe los servicios de la GUI.

El fichero de configuración “applicationContext.xml” es muy sencillo e indica las dependencias entre cada uno de los objetos utilizados en la aplicación:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
      "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
      <bean id="usuarioDao" class="dao.impl1.UsuarioDaoStatic">
      </bean>
      <bean id="interfazRegistro" class="registrar.Registro">
            <property name="dao">
                  <ref bean="usuarioDao"/>
            </property>
      </bean>
</beans>
```

En este ejemplo podemos observar lo fácil que sería sustituir la base de datos estática por una base de datos relacional, simplemente deberíamos crear una clase que implementase la interfaz “UsuarioDao” y en el fichero “applicationContext.xml” indicar que los bean “usuarioDao” son de la nueva clase creada en vez de la clase “dao.impl2.UsuarioDaoStatic2”, con esto nuestra aplicación seguiría funcionando correctamente pero esta vez utilizando una base de datos relacional (o la que se quiera usar). En el caso de lo que deseemos cambiar sea la GUI por otra más elaborada o por una Web el proceso sería semejante e igual de sencillo. Además en este ejemplo contamos con una clase JUnit (“test.TestUsuarioDao”) con la que realizamos un test de la implementación de la base de datos junto con Spring para probar su funcionamiento.

Nota: Para que el proyecto eclipse funcione se deben añadir las librerías de la carpeta “lib” al path.

Ampliación Registro Usuarios: Hibernate

En este ejemplo incluiremos una nueva clase, “dao.Hibernate.UsuarioDaoHibernate”que accederá a una base de datos relacional en vez de las bases de datos estáticas que implementábamos con “dao.impl1.UsuarioDaoStatic” y “dao.impl2.UsuarioDaoStatic2” a través de Hibernate.

Hibernate es un entorno de trabajo que tiene como objetivo facilitar la persistencia de objetos Java en bases de datos relacionales y al mismo tiempo la consulta de estas bases de datos para obtener objetos. Para llevar a cabo esta tarea Hibernate necesita saber los siguientes puntos:

1. Quién está detrás del modelo relacional:

    1) Qué gestor de bases de datos está detrás
    2) A qué base de datos me conecto
    3) Cómo me conecto
    4) Etc.

2. Cómo se emparejan propiedades y campos de tablas:

    - *Clave primaria:*

        1) ¿Cuál es la propiedad que se corresponde con la clave primaria de la tabla correspondiente?
        2) ¿Qué método deberé utilizar para generar un nuevo valor de la clave primaria?
        3) Etc.

    - *Otras propiedades:*

        1) Cómo empareja una propiedad con un campo de una tabla de la base de datos
        2) ¿Cómo se llama la columna correspondiente?
        3) ¿Qué tipo tiene?
        4) ¿Es obligatoria?
        5) Etc.

    - *Cómo gestiona las relaciones entre tablas*

        1) ¿Es una relación “uno-a-uno”,
        2) “uno-a-muchos”,
        3) “muchos-a-muchos”?, etc.

Para poder responder a responder a las dos grandes preguntas se cuentan con dos tipos de archivos: con el archivo de propiedades (Hibernate.properties) y de configuración de Hibernate (hibernate.cfg.xml) que son los encargados de determinar los aspectos relacionados con el gestor de bases de datos y las conexiones con él, y con los archivos que definen el emparejamiento (mapping) de los atributos de nuestras clases con las

tablas y columnas (*.hbm.xml). En nuestro caso Spring nos da soporte para integrar Hibernate, de modo que los archivos anteriores no serán necesarios (salvo los *.hbm.xml) al proporcionar la información necesaria a través del archivo de configuración de Spring (applicationContext.xml).

La configuración de la conexión con Hibernate se la proporcionaremos mediante el bean "myDataSource":

```
<bean id="myDataSource"
      class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close"
      abstract="false" singleton="true" lazy-init="default"
      autowire="default" dependency-check="default" >
      <property name="driverClassName"
            value="org.hsqldb.jdbcDriver" />
      <property name="url" value="jdbc:hsqldb:file:testdb" />
      <property name="username" value="sa" />
      <property name="password" value="" />
</bean>
```

DriverClassName indica el tipo de driver que vamos a utilizar para realizar la conexión con la base de datos, *url* especifica la dirección en la que se ubica y *username* y *password* son el nombre de usuario y contraseña con el que establecemos la conexión.

Las propiedades de la conexión las definimos mediante el bean "mySessionFactory":

```
<bean id="mySessionFactory"
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBea
n"    abstract="false" singleton="true" lazy-init="default"
      autowire="default" dependency-check="default" >
      <property name="dataSource" ref="myDataSource" />
      <property name="mappingResources">
            <list>
                  <value>modelo/Usuario.hbm.xml</value>
            </list>
      </property>
      <property name="hibernateProperties">
            <props>
                  <prop key="hibernate.dialect">
                        org.hibernate.dialect.HSQLDialect
                  </prop>
                  <prop key="hibernate.show_sql">true</prop>
                  <prop key="hibernate.hbm2ddl.auto">create</prop>
            </props>
      </property>
</bean>
```

En este bean los más importante a parte de definir el dialecto con el que nos comunicaremos con Hibernate es la lista con las rutas de todos los archivos *.hbm.xml correspondientes a cada una de las clases que queremos mapear, en nuestro ejemplo sólo una.

La tabla que usaremos en nuestro ejemplo encargado de almacenar los objetos de la clase "modelo.Usuario" se podría crear mediante el siguiente script:

```
CREATE TABLE `USUARIO` (
        `id` INTEGER UNSIGNED NOT NULL,
        `nombre` VARCHAR(45) NOT NULL,
        PRIMARY KEY(`id`)
)
```

Para la tabla del ejemplo el archivo de Hibernate ("Usuario.hbm.xml") que deberemos crear será el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="modelo.Usuario" table="USUARIO">
      <id name="id" column="id" type="integer" unsaved-value="-1">
      <!-- Le proporcionamos nosotros el valor de la clave -->
            <generator class="assigned"/>
      </id>
      <property name="nombre" not-null="true" />
  </class>
</hibernate-mapping>
```

- **<class>**

  Define la relación entre una clase y una tabla, con *name* indicamos el nombre de la clase y con *table* el de la tabla.

- **<id>**

  Empareja uno de los atributos de la clase con la clave primaria de la tabla. *name* es el nombre del atributo y *column* la columna de la tabla a la que se le asocia, además indicamos de que tipo es y su valor cuando aun no se le haya asignado nada (unsaved-value). Para generar el valor de la clave podemos disponer de tres métodos:

  <generator class="**native**"**/>**: Deja que Hibernate escoja entre los métodos identity, sequence o hilo, en función de las características del gestor de bases de datos con el que trabajemos.

  <generator class="**assigned**"/>: Deja que sea nuestra aplicación la que asigne un identificador al objeto antes de guardarlo en la base de datos.

  <generato class="**increment**"/>: Genera identificadores de tipo long, short o int que son únicos sólo cuando no hay otro proceso que esté insertando datos en la misma tabla. Por lo tanto, no es recomendable usarlo, por ejemplo, en un cluster.

<porperty>

Sirve para emparejar aquellos atributos que no forman parte de la clave primaria con las correspondientes columnas de una (o más) tablas.

```
<property name="nombre" column="COLUMNA" type="tipo"
```

unique=”true” **not-null**="true"/>

Type, unique, not-null son opcionales, con ellos indicamos el tipo del atributo, si su valor debe ser único y si permitimos que tome valores nulos respectivamente. Además podemos establecer el tipo de relación (*one-to-one, one-to-many, many-to-many*) entre unas tablas y otras:

<class name="paquete.Clazz" table="clazz">
...
<many-to-one name="foo" column="ID_FOO"/>
...
Indicamos que el atributo foo (que Hibernate sabe que es del tipo Foo) se obtiene a partir del identificador ID_FOO de la tabla CLAZZ.

O como podemos observar en un fragmento de “AssessmentImpl.hbm.xml”:

```
<class table="ASSESSMENT"
      name="es.eucm.assessment.pojo.AssessmentImpl">

  <id type="java.lang.Long" column="ASI_ID" access="field"
  name="id">
  <generator class="native"/>
</id>
<set table="ASSESSMENT_SECTIONS" access="field" name="sections">
  <key column="ASI_ID"/>
  <many-to-many column="SECTION_ID"
      class="es.eucm.assessment.pojo.SectionImpl"/>
</set>
```

Donde se indica que:

- Estamos relacionando la clase es.eucm.assessment.pojo.AssessmentImpl con la tabla ASSESSMENT.

  ```
  <class table="ASSESSMENT"
     name="es.eucm.assessment.pojo.AssessmentImpl">
  ```

- Donde el atributo sections se relaciona con la tabla ASSESSMENT_SECTIONS.

  ```
  <set        table="ASSESSMENT_SECTIONS"        access="field"
  name="sections">
  ```

- La clave primaria de la tabla ASSESSMENT_SECTION que se relaciona con la tabla ASSESSMENT es ASI_ID.

  ```
  <key column="ASI_ID"/>
  ```

- El atributo sections tiene una relación “muchos a muchos” con la columna SECTION_ID de la tabla es.eucm.assessment.pojo.SectionImpl.

  ```
  <many-to-many column="SECTION_ID"
                class="es.eucm.assessment.pojo.SectionImpl"/>
  ```

- El atributo sections contiene objetos del tipo es.eucm.assessment.pojo.SectionImpl, con lo cual, Hibernate tiene elementos suficientes para saber que ha de relacionar el identificador de SectionImpl con la columna SECTION_ID de la tabla ASSESSMENT_SECTION.

Descrito esto, la forma de la que podemos salvar, modificar, buscar o eliminar un usuario es la siguiente:

```
public void saveUsuario(Usuario usuario) {
        this.logger.debug("Intentamos guardar el usuario " + usuario);
        HibernateTemplate temp = getHibernateTemplate();
        if (usuario != null) {
                List listado = temp.find("FROM " + Usuario.class.getName()
                                + " as usuario where usuario.id =" +
                                usuario.getId());
                if (listado.isEmpty()) {
                        this.logger.debug("No lo contiene, hacemos un save");
                        temp.save(usuario);
                } else {
                        this.logger.debug("Lo contiene, hacemos un update");
                        temp.update(usuario);
                }
        }
}

public Usuario findUsuario(Integer id) {
        this.logger.debug("Buscamos el usuario " + id);
        return (Usuario) getHibernateTemplate().get(Usuario.class, id);
}

public void deleteUsuario(Integer id) {
        this.logger.debug("Borramos el usuario " + id);
        Usuario usu = (Usuario) getHibernateTemplate().load(Usuario.class, id);
        getHibernateTemplate().delete(usu);
}
```

Si no queremos utilizar HibernateTemplate podemos utilizar SessionFactory para obtener una sesión con la que podamos realizar las transacciones como podemos observar en el siguiente ejemplo:

```
public class ProductDaoImpl implements ProductDao {

   private SessionFactory sessionFactory;

   public void setSessionFactory(SessionFactory sessionFactory) {
      this.sessionFactory = sessionFactory;
   }

   public Collection loadProductsByCategory(String category) {
      return this.sessionFactory.getCurrentSession()
```

```
            .createQuery("from test.Product product where product.category=?")
            .setParameter(0, category)
            .list();
    }
}
```